NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

ADA IMPLEMENTATION OF CONCURRENT EXECUTION OF MULTIPLE TASKS IN THE STRATEGIC AND TACTICAL LEVELS OF THE RATIONAL BEHAVIOR MODEL FOR THE NPS *PHOENIX* AUTONOMOUS UNDERWATER VEHICLE (AUV)

by

Michael John Holden

September 1995

Thesis Advisors: Robert B. McGhee
Man-Tak Shing

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE September 1995	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE ADA IMPLEMENTATION OF CONCURRENT EXECUTION OF MULTIPLE TASKS IN THE STRATEGIC AND TACTICAL LEVELS OF THE RATIONAL BEHAVIOR MODEL FOR THE NPS PHOENIX AUTONOMOUS UNDERWATER VEHICLE(AUV)	5. FUNDING NUMBERS
6. AUTHOR(S) Holden, Michael John	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Current autonomous vehicle control systems are limited to inefficient sequential operation because of a lack of concurrency in program execution. When one segment of a sequential control system is delayed or fails, the remaining segments cannot be executed unless extensive error-handling routines are invoked. Undersea robotic vehicles in particular are subject to potential catastrophic loss in the event of minor program faults. The problem addressed by this thesis is to provide concurrent execution and higher level abstraction in reliable and maintainable control software, specifically for the Naval Postgraduate School's Phoenix Autonomous Underwater Vehicle (AUV) and within the framework of the Rational Behavior Model (RBM) Strategic and Tactical Levels. The approach taken for this work was to design and implement the RBM Strategic and Tactical level control software using Ada. The program design adds concurrency using the Ada task construct to create objects that perform separate operations simultaneously. For comparison purposes, the same functionality in a non-concurrent program was also implemented in the LISP programming language. The result was a concurrent RBM control software system in Ada, including a mission planner and mission controller, which can replace the existing sequential mix of software. This work was validated through successful man-in-the-loop simulation of several behavioral doctrines modeling the interaction found aboard manned submarines.

14. SUBJECT TERMS Autonomous vehicle, robot, AUV, Rational Behavior Model, RBM, Phoenix, real-time, concurrency, Ada, control software	15. NUMBER OF PAGES 165
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

ADA IMPLEMENTATION OF CONCURRENT EXECUTION OF MULTIPLE TASKS IN THE STRATEGIC AND TACTICAL LEVELS OF THE RATIONAL BEHAVIOR MODEL FOR THE NPS *PHOENIX* AUTONOMOUS UNDERWATER VEHICLE (AUV)

Michael John Holden
Commander, United States Navy
B.S., United States Naval Academy, 1978

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1995

Author: [illegible]
Michael John Holden

Approved by: [illegible]
Robert B. McGhee, Thesis Co-Advisor

[illegible]
Man-Tak Shing, Thesis Co-Advisor

[illegible]
Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

Current autonomous vehicle control systems are limited to inefficient sequential operation because of a lack of concurrency in program execution. When one segment of a sequential control system is delayed or fails, the remaining segments cannot be executed unless extensive error-handling routines are invoked. Undersea robotic vehicles in particular are subject to potential catastrophic loss in the event of minor program faults. The problem addressed by this thesis is to provide concurrent execution and higher level abstraction in reliable and maintainable control software, specifically for the Naval Postgraduate School's *Phoenix* Autonomous Underwater Vehicle (AUV) and within the framework of the Rational Behavior Model (RBM) Strategic and Tactical Levels.

The approach taken for this work was to design and implement the RBM Strategic and Tactical level control software using Ada. The program design adds concurrency using the Ada *task* construct to create objects that perform separate operations simultaneously. For comparison purposes, the same functionality in a non-concurrent program was also implemented in the LISP programming language.

The result was a *concurrent* RBM control software system in Ada, including a mission planner and mission controller, which can replace the existing sequential mix of software. This work was validated through successful man-in-the-loop simulation of several behavioral doctrines modeling the interaction found aboard manned submarines.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

Many thanks to my patient and loving wife, Cindy, who has been a great source of strength all through this work.

Thanks also to Professor McGhee and Professor Shing, for letting me proceed my own way until I needed some mid-course guidance, and for being right there with the thrusters when asked!

Thanks also to my classmates in CS41 and members of the AUV Research Group for the friendship and energy expended in extra-curricular activities, encouragement and comic relief when things started to get crazy.

I. INTRODUCTION

A. BACKGROUND

This thesis focuses on three areas of the software control systems of autonomous underwater vehicles (AUVs): concurrent execution, higher level abstraction, and reliable, maintainable control software.

1. Concurrent Execution

Autonomous vehicles -- generally defined as vehicles that are capable of intelligent motion and action without requiring either a guide to follow or teleoperator control[1] -- need stable control systems for their hardware level and at the same time require the ability to make high-level logical decisions based upon circumstances and sensor inputs.

Using the paradigm of a manned submarine, even if the Commanding Officer is still in the process of deciding what strategic action to order next, and even if the Officer of the Deck is busy carrying out the skipper's standing orders regarding safe navigation, the control of the ship's propulsion and maneuvering systems must be continuous. These three activities must be able to proceed concurrently.

When autonomous vehicles are operated in an underwater environment, the need for a continuously operating hardware control system becomes more critical. Should all hardware control stop on a land vehicle, in most circumstances the vehicle (or, more unhappily, the wreckage) is recoverable, as in the case of the *Dante II* vehicle at Alaska's Mount Spurr volcano in 1994. [2]

Even with an aerial vehicle, a parachute recovery system can usually allow recovery and further analysis to determine the cause of the fault. However, with underwater vehicles, the loss of the control system frequently results in the complete loss of the vehicle, preventing even post-crash analysis to help discover and correct the reasons for the loss. Thus, for autonomous underwater vehicles particularly, there is a crucial need for concurrent operation of the hardware control systems and the behavioral control systems - those used by the vehicle to determine what higher level actions to take.

2. Abstraction at the Mission Specialist Level

For an autonomous vehicle system to be useful in practical missions, a high level of abstraction in mission planning is needed. The mission specialist should not be burdened with the need to comprehend the subtleties of backtracking or program conditional loops; his task is to provide the mission for the vehicle. This involves specifying the mission track, including start and recovery points; the type and order of phases desired; actions in the event of unsuccessful completion of each phase; and whether specific abort conditions result in a complete mission abort or merely abort applicable phases in the overall mission.

3. Maintainability and Reliability

For any multidisciplinary project involving computer software, the need for maintainability and reliability of the software has proven to be critical if the project is to continue for any significant time span. In addition, modularizing the software can produce significant cost savings if portions of the code can be reused.[3]

These three considerations - concurrent execution, higher level abstraction, and reliable, maintainable control software - for control of autonomous vehicles in general, and more specifically for control of the NPS Autonomous Underwater Vehicle, provide the major impetus for this research.

B. PROBLEM STATEMENT

The objective of this thesis is to develop a reliable and maintainable software system for the concurrent execution of tasks in autonomous vehicle control, specifically within the framework of the Rational Behavior Model (RBM) Strategic and Tactical levels. It is also desired that sufficient abstraction be achieved at the Strategic Level in order that the mission specialist needs the minimum possible programming language knowledge.

C. SCOPE

The primary goal of this research is to develop and validate an implementation of the Strategic and Tactical levels of the RBM in Ada, and compare this with other working implementations.

D. THESIS ORGANIZATION

Chapter II surveys previous work on concurrent execution, work involving Ada in autonomous vehicle control systems, and work at NPS concerning the Autonomous Underwater Vehicle (AUV) *Phoenix* and the Rational Behavior Model. Chapter III presents arguments for object oriented programming, encapsulation of routines, and modular design of the AUV software. Chapter III also discusses the principles and goals of software engineering and the benefits in maintainability and revisability that Ada provides as features of the language. In Chapters IV and V, the Strategic and Tactical level implementation is provided in detail. Chapter VI presents a discussion of testing and the results obtained. Chapter VII provides a summary, conclusions, and recommendations for future research.

II. PREVIOUS WORK

A. INTRODUCTION

In real-time applications, the correctness of computation depends not only on the results of the computation but also on the timeliness of the response. A late answer, when obtained in a system with real-time constraints, is a wrong answer, despite being correct computationally. Examples of real-time applications are nuclear power plant controls, avionics, air traffic control, weapons systems and sensors, and autonomous vehicle control systems.

There are two predominant approaches to current programming efforts in real-time systems. The traditional method is the use of a cyclical executive - sometimes referred to as the "one big loop" approach, wherein the programmer has to lay out the timeline by hand to ensure the successful serial execution of critical sections and deadlines. There is a widely held view that this method becomes unmanageable as the size and complexity of a system increases.[4]

The Ada tasking approach represents a second, fundamentally different model. Ada includes the *task* construct that allows management of concurrent processes at a more abstract level while the Ada run-time system handles the details of task executions. While this approach makes predictability and proof of deadline satisfaction difficult, it is not impossible, and this subject is continuing to be explored theoretically. For periodic tasks, the scheduling problem has been completely solved using rate monotonic scheduling theory.[4]

Real-time robotic applications implementing concurrency using multitasking are increasing in frequency, and general enthusiasm for the use of Ada is likewise rising as programmers gain experience and managers observe the results that are achieved using Ada. The following projects all employed Ada or compared the use of Ada to the use of other languages in developing real-time systems.

B. CalTech/JPL Steler

The Supervisory Telerobotics Laboratory (Steler) at the Jet Propulsion Laboratory (JPL) at the California Institute of Technology has developed a prototype telerobotic system intended for use aboard NASA's Space Station Freedom. The Modular Telerobot Task Execution System (MOTES), written entirely in Ada, monitors and transmits sensor data and provides real-time control of the robot.

The JPL development team deliberately chose Ada as the programming language because of Ada's use of generics, task constructs for asynchronous action, modularity, and portability of code. Ada proved sufficiently flexible to handle real-time robotic applications, and the development team expects that all newly developed software for the space station will be done in Ada. Using Ada significantly shortened the time to develop the prototype, the project team believes, because the complete specification of interfaces which the language requires caused design errors to be caught very early in the development process.

Of particular interest, even though the MOTES system was designed for space station telerobotics, it has additional applications in undersea operations and demonstrated the feasibility of local-remote real-time robotic applications using Ada.[5]

C. NASA OMV

NASA's Orbital Maneuvering Vehicle (OMV) is a semi-independent spacecraft designed to provide services such as delivery, retrieval and reboosting to other spacecraft. The craft has automatic navigation and rendezvous capabilities, but requires human interaction for terminal docking with the space station. Control can be provided from the space shuttle, a ground station, or ultimately from the space station. The OMV can carry various mission kits and can remain in orbit for nine months before requiring maintenance or refuelling.

The software for the OMV was prototyped using Ada. However, tasking was not used due to concerns about the system's strict real-time scheduling requirements[6].

D. INTELLIGENT AVIONICS APPLICATIONS

Since 1988, the Software Systems Group of Boeing's Military Airplane Division has been re-engineering a number of knowledge-based systems for intelligent mission avionics applications. LISP-prototyped implementations are being converted to Ada and evaluated in a laboratory simulation environment to determine the viability of the Ada implementations. The group characterizes future air crew decision aids as providing real-time decision support in rapidly evolving situations and adaptively assisting the human pilot and aircrew in a cycle of sensing-assessing-planning-decision-actions. This same cycle can be adapted to virtually any autonomous vehicle.

From this group's experience, "without much attention to dynamic memory management issues during translation [from LISP to Ada], the Ada version gained an impressive speedup." Since most of this group's prototype applications exhibit potential for using parallelism, they are planning to place more emphasis on using Ada tasking (which was avoided earlier to simplify program conversions.) This group concludes that the "future of intelligent avionics software largely depends on our ability to create, in Ada, adaptive and dynamic software that is reliable, verifiable, and maintainable."[7]

E. NPS AUV *Phoenix*

The Naval Postgraduate School's Autonomous Underwater Vehicle (AUV) Research Group has been working for several years with the *Phoenix* AUV. [8]

1. Vehicle Description

Essentially a robot minisubmarine, *Phoenix* is a seven-foot-long, four hundred pound unmanned untethered vehicle. Designed as a testbed for research in underwater vehicle control systems as well as in robotic software control logic, *Phoenix* is the focus of ongoing research work for Ph.D. and M.S. students in Computer Science, Mechanical Engineering, Electrical and Computer Engineering, Undersea Warfare, and other disciplines at the Naval Postgraduate School.

2. Software Control

Software control of *Phoenix* was designed to be in accordance with the Rational Behavior Model (RBM), more fully described below.[9] As development proceeded, the patches for program problems and the lack of a well-defined strategic and tactical level has somewhat blurred the dividing lines between the three levels described in RBM. The resulting code is workable but hard to maintain and harder still for a new user to decipher. The current working version as implemented in the vehicle is a combination of PROLOG for the Strategic level and C for the execution level, with the tactical level behaviors a combination of the two. A change in the mission often requires reworking both the strategic and tactical level code.

3. Tactical Level Implementation in Ada

Byrnes[9] used Classic-Ada (an extension of the standard Ada language) to obtain Object-Oriented Programming in his simulation of the Tactical Level. Thornton[10] developed and evaluated a concurrent, object-based implementation in Ada for the Tactical level of the RBM. Using a combination of relay tasks for intra-object concurrency and parameterless task entry calls, his simulation testing validated Ada's ability to ensure continuous control of the vehicle at the Execution level while performing simultaneous time-consuming tasks at the Tactical level. His conclusion was that Ada was the programming language of choice for implementation of the Tactical Level, despite a lack of true object-oriented features, due to its ability to handle concurrency.

F. UNDERWATER VIRTUAL WORLD

Brutzman[11] designed, developed and implemented a three-dimensional distributed "virtual world" that allows users across the internet to see visually the results of execution-level commands sent to the NPS AUV and other vehicles. This simulation environment allows use of various environmental and vehicle response models and is designed to allow full testing of AUV software designs prior to the expense of implementation in an undersea vehicle.

G. THE RATIONAL BEHAVIOR MODEL

1. Introduction

The Rational Behavior Model (RBM), first discussed by Kwak, McGhee and Bihari[12], and developed in detail by Byrnes[9], grew out of a strong interest among researchers in robotics and artificial intelligence in linking high level logical mission planning for autonomous vehicles with low level vehicle control programming. Their efforts resulted in a three-level software architecture consisting of the Strategic, Tactical and Execution levels with respective emphasis on mission planning, programmed vehicle responses labeled "behaviors," and efficient real-time execution of vehicle hardware control programming. Figure 1 illustrates the relationships between the three levels of RBM.

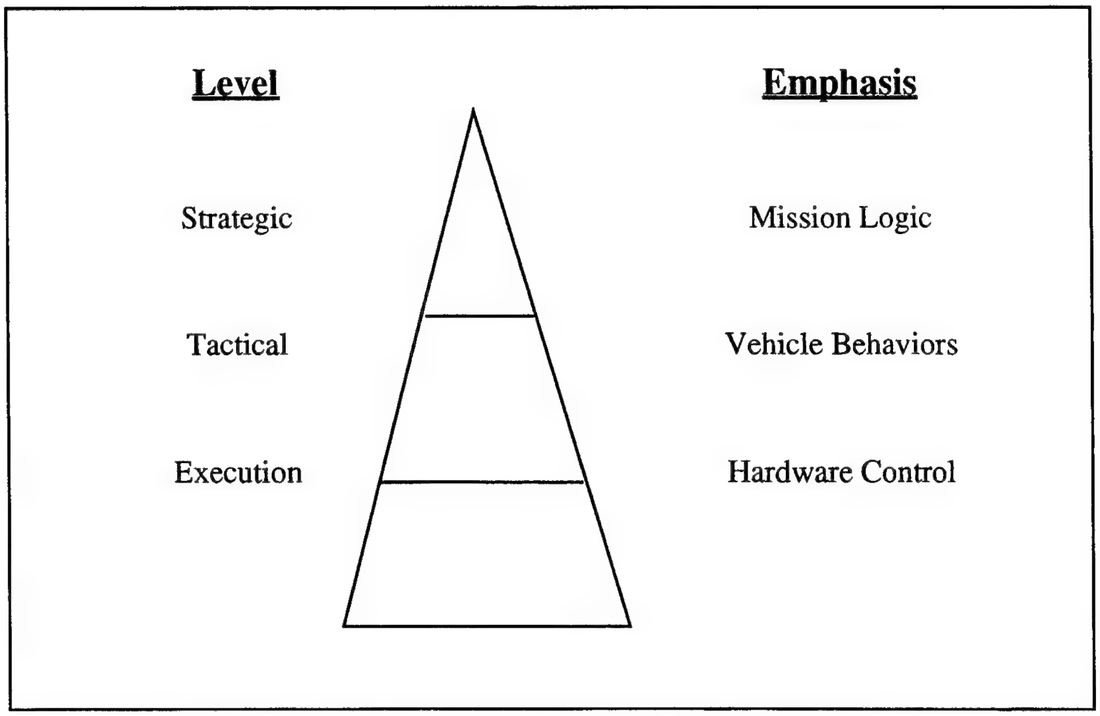


Figure 1. RBM Architecture Levels and Emphasis

2. NPS AUV Implementation of the RBM

The NPS AUV implementation of the RBM was chosen to provide a well-known and well-understood model for the relationships between the various levels - that of the human interaction that occurs in the command structure of manned submarines. One serendipitous result of using this paradigm is that it clearly reveals the concurrent actions of disjunct objects that should occur for efficient operation. Figure 2 illustrates this equivalence for the NPS AUV model of the RBM.

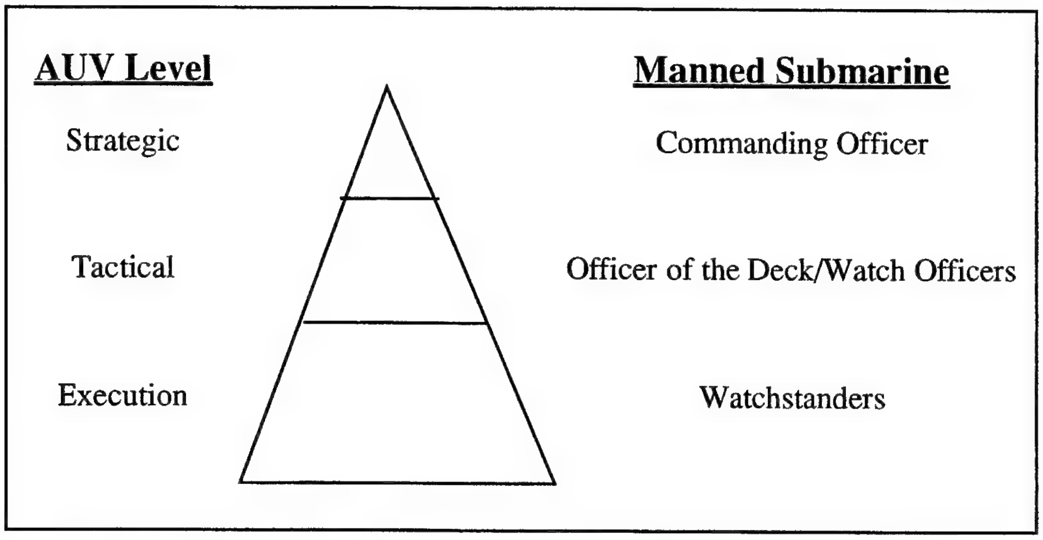


Figure 2. NPS AUV RBM Levels and Symbolic Equivalences

a. Strategic Level

At this level, high-level mission logic provides for the deterministic sequencing of the underlying behaviors implemented for that particular vehicle. In the manned submarine, this is the Commanding Officer.

b. Tactical Level

At this level, programming implements the behaviors capable of satisfying the goals assigned by the Strategic level and provides a symbolic-to-numeric interface for issuing the commands necessary to direct the performance of the Execution level. The manned submarine equivalent is the Officer of the Deck who coordinates the ship's watch officers in satisfying the Commanding Officer's high-level orders (goals).

c. Execution Level

Responsible for all of the physical actions of the vehicle, this level of the software architecture is the intermediary between the Tactical level and the actual hardware of the vehicle, and must meet all the hard real-time scheduling requirements to ensure basic vehicle stability. In the manned submarine paradigm, this level represents the specific crewmembers on watch maintaining the ships's maneuvering, navigation, propulsion, and similar systems.

H. SUMMARY

Previous work in real-time autonomous vehicle control has shown the feasibility and advantages of using the object-oriented, multitasking capabilities of the Ada programming language. General enthusiasm for use of the Rational Behavior Model's three-level architecture is expanding as researchers gain experience with the paradigm.

III. SOFTWARE ENGINEERING ISSUES

A. INTRODUCTION

The continuing rapid development of computer hardware and software has made it possible to attempt to solve real world problems that were previously out of reach of computer automation. At the same time, this development has raised general expectations of performance. Consequently, those higher expectations have required far more complex solutions than were previously attainable. As early as 1972, Dijkstra stated that the capability of hardware systems had far exceeded our ability to manage them[13]. Booch refers to this change in the software development process from a labor-saving activity to a labor-intensive one as the *software crisis*.[3]

To resolve this problem and manage the complexity of large software systems, the concept of a disciplined approach to software development has risen, complete with concrete goals and principles.

B. SOFTWARE ENGINEERING GOALS AND PRINCIPLES

1. Goals

Stated requirements even for small systems are frequently incomplete when they are originally specified. Besides accomplishing these stated requirements, a good software system must be able to easily support changes to these requirements over the system's life. Thus, a major goal of software engineering is to be able to deal with the effects of these changes. The most often cited software engineering goals are:

a. Maintainability

It should be possible to easily introduce changes to the software without increasing the complexity of the original system design.

b. Reliability

The software should prevent failure in design and construction as well as recover from failure in operation. Put another way, the software should perform its intended function with the required precision at all times.

c. Efficiency

The software system should use the resources that are available in an optimal manner.

d. Understandability

The software should accurately model the view the reader has of the real world. Since code in a large, long-lived software system is usually read more times than it is written, it should be easy to read at the expense of being easy to write, and not the other way around.

2. Principles

In order to attain a software system that satisfies the above goals, sound engineering principles must be applied throughout development, from the design phase to final fielding of the system. These include:

a. Abstraction

"The essence of abstraction is to extract essential properties while omitting inessential detail." [14] The software should be organized as a ladder of abstraction in which each level of abstraction is built from lower levels. The code is sufficiently conceptual so the user need not have a great deal of technical background in the subject. The reader should be able to easily follow the logical path of each of the various modules. The decomposition of the code should be clear.

b. Information Hiding

The code should contain no unnecessary detail. Elements that do not affect other portions of the system are inaccessible to the user, so that only the intended operations can be performed. There are no "undocumented features".

c. Modularity

The code is purposefully structured. Components of a given module are logically or functionally dependent.

d. Localization

The breakdown and decomposition of the code is rational. Logically related computational units are collected together in modules.

e. Uniformity

The notation and use of comments, specific keywords and formatting is consistent and free from unnecessary differences in other parts of the code.

f. Completeness

Nothing is blatantly missing from any module. All important or relevant components are present both in the modules and in the overall system as appropriate.

g. Confirmability

The modules of the program can be tested individually with adequate rigor. This gives rise to a more readily alterable system, and enables the reusability of tested components.

C. COMPONENT REUSE AND AUTOMATED SOFTWARE TOOLS

As the world's largest user of embedded computer systems, the United States Department of Defense (DoD) concluded in 1975 that most DoD software was nonresponsive to user needs, unreliable, excessively expensive, untimely, inflexible, difficult to maintain, and not reusable.[15] The Ada programming language was developed as a major consequence of DoD's growing awareness of these problems. With the advent of promising automated software tools and the ability to store and reuse previously tested components, software design and construction is moving, like many fields before it, from the domain of adepts and artists to the domain of engineers and mechanics.

The Computer Aided Prototyping System (CAPS) at the Naval Postgraduate School (NPS) is designed as a rapid prototyping system for hard real-time systems and uses atomic Ada reusable components and automated and semi-automated software tools.[16] As such, CAPS is ideally suited for use in further work on the Ada version of the control software system of the NPS AUV.

D. COMPARISON OF PROGRAMMING LANGUAGES

A number of evaluations have been made of the use of various programming languages for real-time systems. In the case of object-oriented languages, Ada has been repeatedly found fully capable and in many respects more desirable than other existing languages. LISP, C++ and Ada are all American National Standards Institute (ANSI) Standards. However, this is a new event for C++, and the numerous variations on the language are not likely to be brought into uniformity any time soon. In LISP, "garbage collection" of memory space that is no longer in use can cause unpredictable delays. Ada, on the other hand, has no memory allocation issues, and has been an ANSI standard since its inception in 1983.[18] Ada-95 is the new ANSI standard (ANSI-1815A-1995). Ada has also been an International Organization for Standardization Standard since 1987[19] and is a Federal Information Processing Standard as well[20].

Another approach is being taken in European designs that mesh synchronous and asynchronous programming languages. These designs take advantage of the ability to make part of the overall programming deterministic, allowing use of software tools to perform correctness proofs on the synchronous portions of the program. Synchronous programming requires care in implementation, and in any event cannot serve as a stand-alone solution to real-time programming, as the real world is not synchronous.[21]

Thornton[10], Harrison and Moulding[22] and Steele and Backes[5] all have concluded that the use of Ada for system development is more than feasible and frequently is far preferable to other languages.

In one of the few times a manager has performed the same project several times using similarly experienced teams but with different programming tools, the use of Ada significantly boosted project success rates. At SUNY Plattsburgh, Professor John McCormick has assigned the same real-time programming project to his class for nine years.

> Working in teams of three or four, McCormick's students must write 15,000 lines of code to control a system that would need about 150 switches to operate using hardware alone. In the five years students used C, no team completed the project -- even when more than half of the code was provided. With Ada, however, half of the teams completed the project before any support code had even been written. With some support code now provided, three out of four teams finish the project.
>
> Specific factors in this improvement, according to McCormick, include both syntax and semantics. Ada leaves less room for single-keystroke errors; its type-abstraction facilities reduce the need for error-prone pointer manipulation; and its modular facilities improve teams' coordination of effort. [23]

The Secretary of Defense has reaffirmed DoD's commitment to using Ada, regardless of size, cost, or functional application, for all new projects and major modifications to existing systems unless Commercial Off-The-Shelf (COTS) products will suffice to do the job[24][25]. In March 1995, the revised Ada-95 became Federal Information Processing Standard 191-1, and Ada-95, like the previous version (now called Ada-83) is now both a national (ANSI) and international (ISO) standard. Use of Ada in defense systems is also mandated by law[26].

A survey of Department of Defense (DoD) programming languages in 1995 reveals that Ada is the language of choice among weapons systems programmers and the second choice (behind COBOL) among information systems developers. The survey vindicates Ada supporters who have stuck with the language despite massive criticisms from various quarters. The survey also notes that of DoD's at least 450 languages and various mutations used since the 1970s, only 37 of those are still useful in 1995. DoD officials say that the Ada mandate continues to flourish because it is cutting down the rising cost of software maintenance.[27]

E. SUMMARY

Given the large body of evidence that the language is capable of handling the real-time and concurrent programming issues involved, and recognizing that the DoD mandate still calls for Ada to be used for interoperability, Ada was the choice of programming language for this thesis. In the process, the LISP programming language was used for a comparable non-concurrent implementation of the Strategic and Tactical Levels of the RBM, since a large number of knowledge-based systems for intelligent mission applications have been prototyped in that language.

IV. ADA IMPLEMENTATION OF RBM IN *Phoenix*

A. INTRODUCTION

Previous implementations of the Rational Behavior Model (RBM) for the NPS AUV *Phoenix* have used the natural model of the hierarchy of personnel manning for the watch stations in a manned submarine. However, these implementations have been limited because of the sequential nature of the implementing language. The use of an object-oriented language such as LISP or Ada will increase flexibility in missions by allowing relatively simple exchange of Tactical Level objects for specific missions without requiring complete rewriting of the tactical level code. In addition, the paradigm of the manned submarine crew can be more accurately represented by executing the concurrent activities of the various watch officers through the Ada programming language's *task* constructs. While still providing tight control of the execution of behaviors in the Tactical Level by the Officer of the Deck (OOD) Object, routine activity required by subordinate objects to efficiently complete their assignments is allowed to proceed in parallel.

B. STRATEGIC LEVEL OVERVIEW

At this level, high-level mission logic provides for the deterministic sequencing of the underlying behaviors implemented for the vehicle. In the manned submarine, this is the Commanding Officer's role.

This level of the RBM architecture is the most abstract, allowing the Mission Specialist to understand only a minimum level of programming. There is a set of questions the Mission Specialist (MS) must answer in setting up a mission:

1. What is the mission path, including the Launch and Recovery Points?
2. What is the order of phases desired in the mission?
3. What action is desired following unsuccessful completion of any phase?
4. Will specific abort conditions result in a complete mission abort or merely abort applicable steps in the mission?

A typical Mission Scenario for the NPS AUV *Phoenix* is shown in Figure 3. *Phoenix* is first initialized and then transits to several locations in the order specified earlier by the Mission Specialist. At each location the AUV performs a search and a task if a target is located during the search. It then transits to a final point for recovery. For each phase the Mission Specialist must delineate actions for the AUV to take if a systems failure occurs.

C. CURRENT STATE OF THE STRATEGIC LEVEL IN *Phoenix*

The Strategic level is currently implemented in the PROLOG logical programming language. A typical example of this code is in Appendix A.

D. TACTICAL LEVEL OVERVIEW

At this level, the concept of vehicle *behaviors* becomes important. Programming implements the actions capable of satisfying the goals assigned by the Strategic Level and provides an interface for issuing the commands necessary to direct the performance of the Execution Level. The manned submarine equivalent is the Officer of the Deck (OOD) who coordinates the actions of the ship's watch officers in satisfying the Commanding Officer's (CO's) high-level orders (goals).

As depicted in Figure 3, the Tactical Level contains the code responsible for how the vehicle conducts a waypoint-following transit, its method for performing a search, and the way it executes a task. These behaviors are best implemented in an object-oriented programming language, where the interfaces to each behavior/object are tightly specified. This allows the use of a library of behaviors, enabling the Mission Specialist to select the specific behavior for the mission from a group of similar behaviors. For example, in a mission to map a mined harbor approach a highly specialized sonar mapping program might be desired, while in a mission to obtain oceanographic samples a much less detailed sonar search behavior would be acceptable. Both the LISP and Ada programming languages support this modularization of code for simple interchangeability of objects.

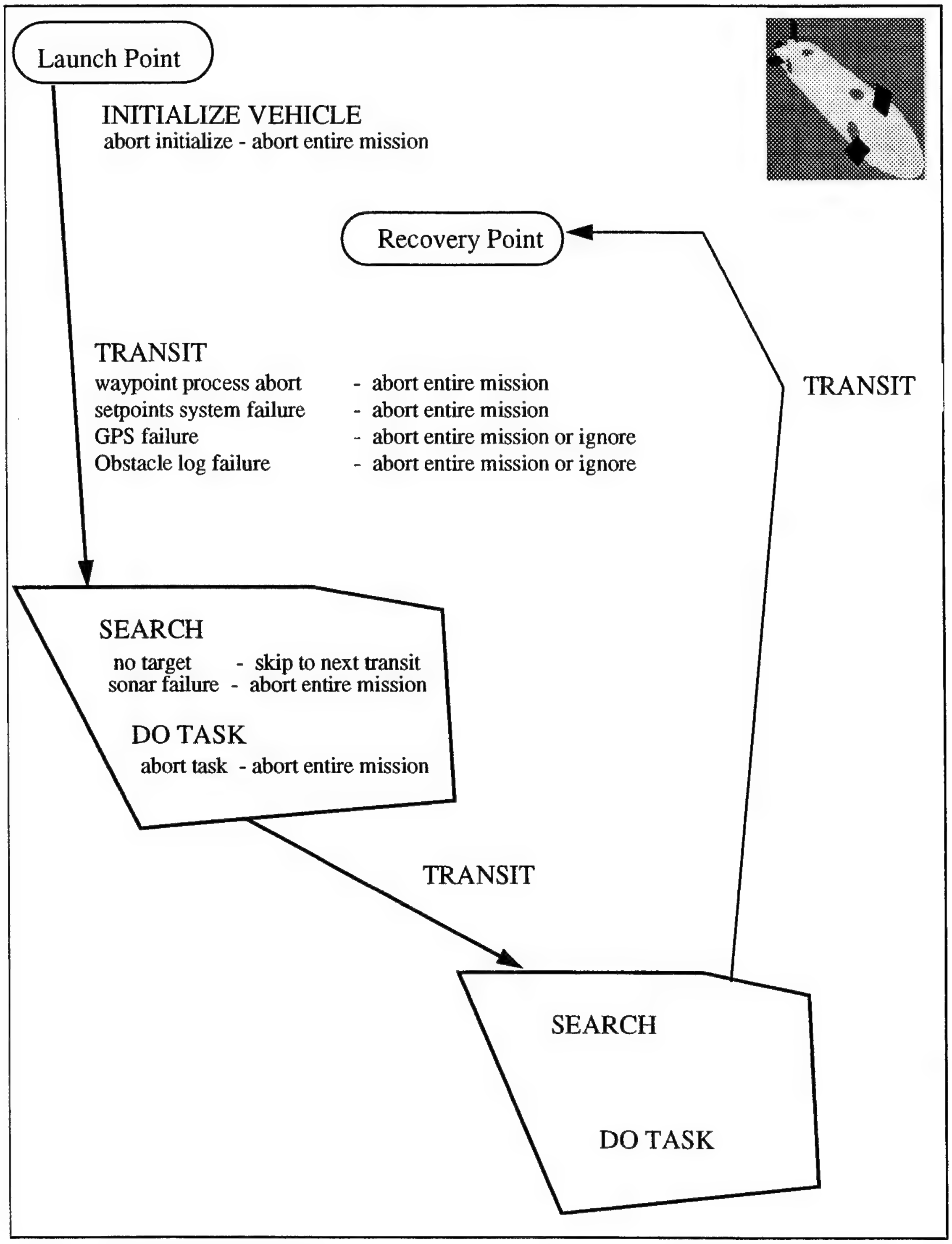

Launch Point
INITIALIZE VEHICLE
abort initialize - abort entire mission
Recovery Point
TRANSIT
waypoint process abort - abort entire mission
setpoints system failure - abort entire mission
GPS failure - abort entire mission or ignore
Obstacle log failure - abort entire mission or ignore
TRANSIT
SEARCH
no target - skip to next transit
sonar failure - abort entire mission
DO TASK
abort task - abort entire mission
TRANSIT
SEARCH
DO TASK

Figure 3. Sample AUV Mission

E. CURRENT STATE OF THE TACTICAL LEVEL IN *Phoenix*

This level is currently implemented in the C programming language. A typical example of this code is shown in Appendix B.

F. LISP DESIGN MODEL

Figure 4 depicts the LISP design model used to implement the Strategic and Tactical Levels of the RBM in a sequential structure.

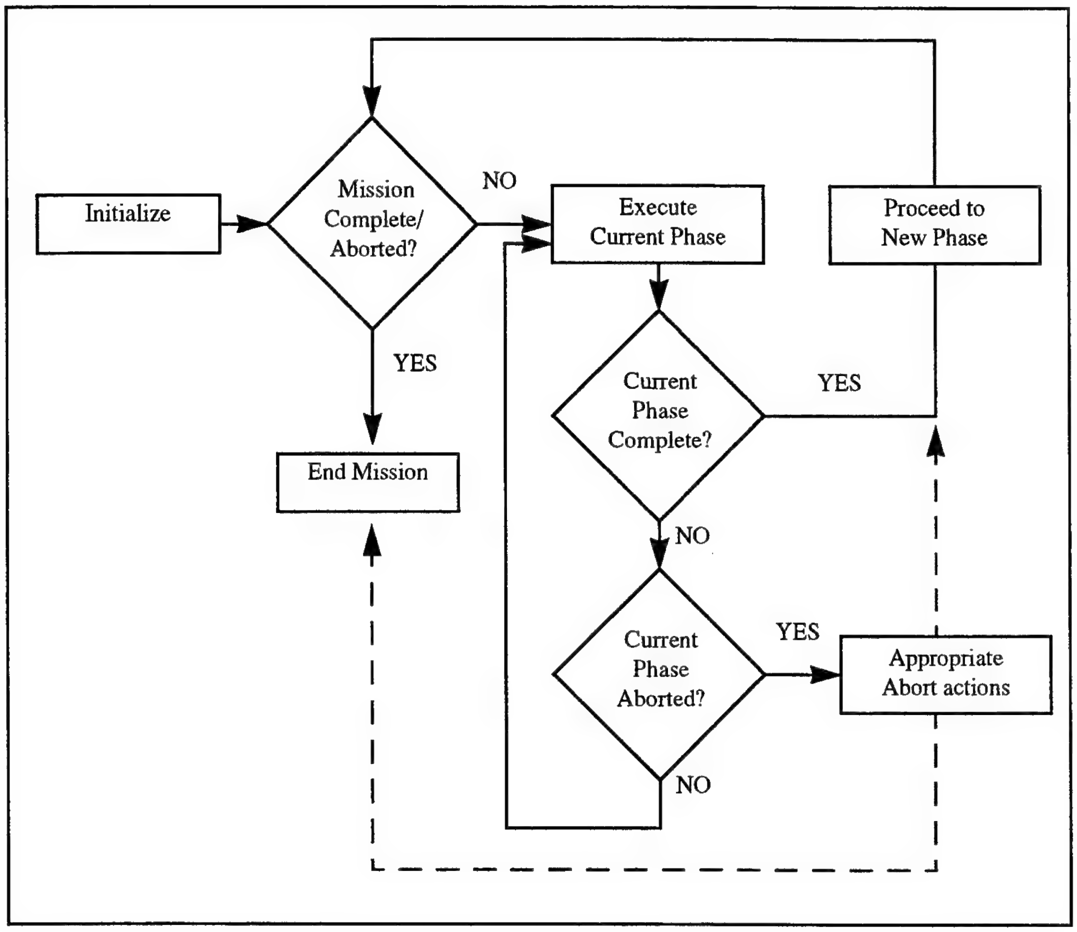
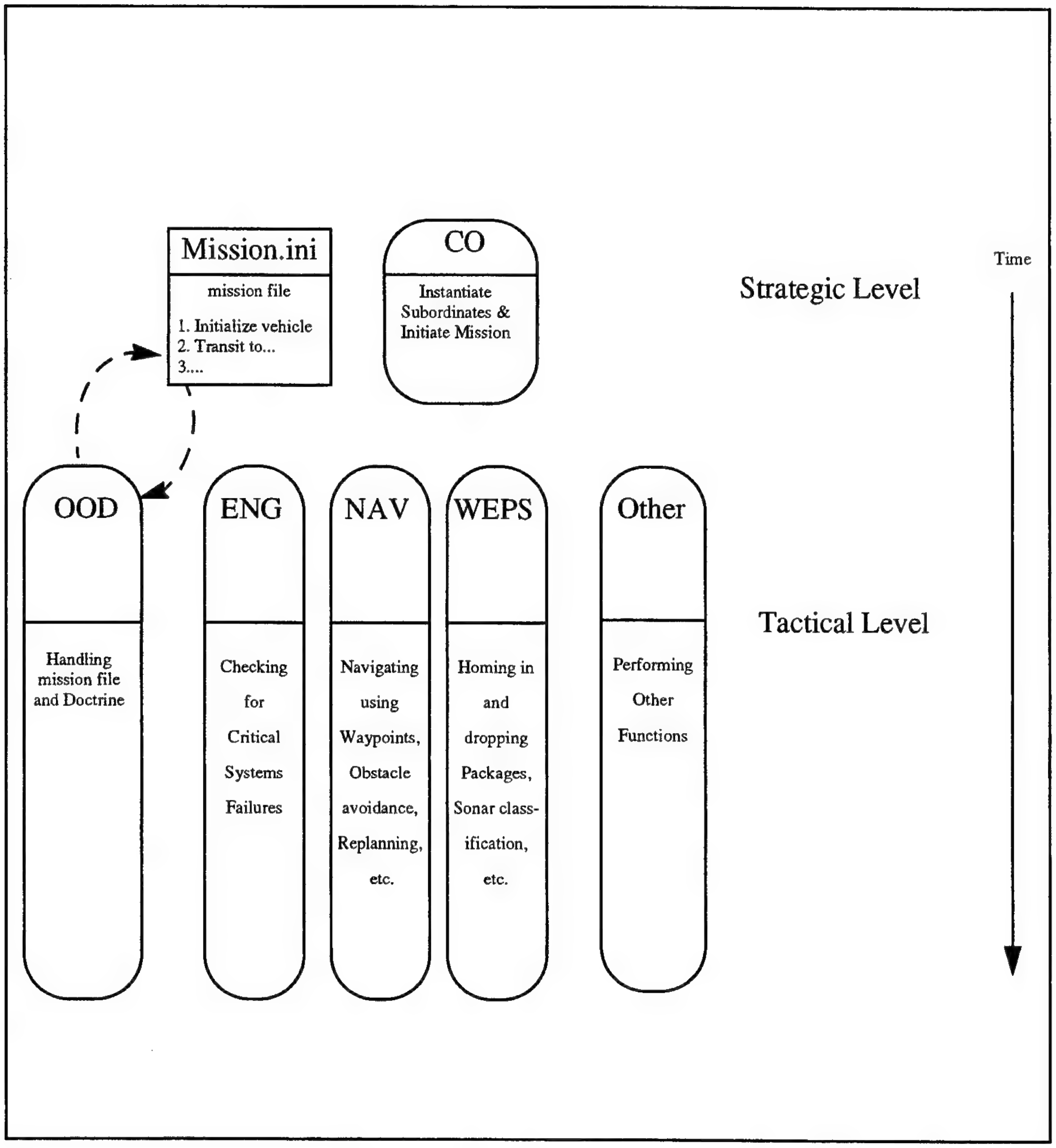


Figure 4. LISP Design Model

As can be seen from examining the LISP design model, this is an example of the cyclical executive or "one big loop" approach to multiple event programming. There is no clear delineation of the Strategic and Tactical Levels. The phases to be executed are listed in the code (see Appendix C) rather than in a separate mission file, requiring more than trivial knowledge of the programming language by the Mission Specialist to ensure satisfactory execution of the program.

G. ADA DESIGN MODEL

Figure 5 depicts the Ada design model used to implement the Strategic and Tactical Levels of the RBM in a concurrent structure.



Mission.ini
mission file
1. Initialize vehicle
2. Transit to...
3....
CO
Instantiate Subordinates & Initiate Mission
Strategic Level
Time
OOD
Handling mission file and Doctrine
ENG
Checking for Critical Systems Failures
NAV
Navigating using Waypoints, Obstacle avoidance, Replanning, etc.
WEPS
Homing in and dropping Packages, Sonar classification, etc.
Other
Performing Other Functions
Tactical Level

Figure 5. Ada Design Model

H. ADA IMPLEMENTATION

The implementation of this model employs the Ada *task* construct to produce multiple objects operating concurrently, shown in Figure 5 as OOD (Officer of the Deck), ENG (Engineer Officer), NAV (Navigator), WEPS (Weapons Officer), and other objects at the Tactical Level. On a single-processor machine, these multiple processes would be run by the CPU in the method called time-slicing, with each process getting a share of the CPU time in sequence, with its individual information being stored and retrieved as needed. Time-slicing for practical purposes allows these processes to run nominally as "concurrent" but, in fact, the CPU is task-switching. This operation swaps each process completely in and out of memory so that only one is running at one time. On a multiple-processor machine, each CPU could run separate processes, thus providing truly parallel operation. If more processes than CPUs are in use, then each CPU could time-slice processes, and still provide a good measure of concurrency.

The dotted line arrows between the OOD object and the mission file indicates that the specifics of the mission can be provided to the OOD object as a compiled file rather than as a complex interaction between communicating processes. In the process of executing a mission, following specific behaviors chosen by the Mission Specialist when constructing the mission file, the Tactical Level objects might need the flexibility to modify the mission file. For example, an additional set of waypoints might be required to be added by the Navigator in the event the vehicle encounters a previously undiscovered obstacle in its path.

A sample mission file as a Mission Specialist would write it is shown in Figure 6.

```
Phase  Action                      Success  Fail Step Type
-------------------------------------------------------
 1     Initialize vehicle           2       98    1
 2     Transit to task location     3       2     2
 3     Search area for target       4       3     3
 4     Report targets found         5       6     4
 5     Commence task on target      6       5     5
 6     Transit to task location     7       6     2
 7     Search area for target       8       7     3
 8     Report targets found         9       10    4
 9     Commence task on target      10      9     5
 10    Transit to recovery point    99      10    11
 98    Abort Mission(98)            98      98    12
 99    Mission Complete(99)         99      99    13
```

Figure 6. File as reviewed by a Mission Specialist

The screen presentation of the same sample mission file is shown in Figure 6

```
Phase     Action                     Next Phase to go to
---------------------------------------------------------------
 1      Initialize vehicle          If Succeed/Fail = 2/ 98
 2      Transit to task location    If Succeed/Fail = 3/ 2
 3      Search area for target      If Succeed/Fail = 4/ 3
 4      Report targets found        If Succeed/Fail = 5/ 6
 5      Commence task on target     If Succeed/Fail = 6/ 5
 6      Transit to task location    If Succeed/Fail = 7/ 6
 7      Search area for target      If Succeed/Fail = 8/ 7
 8      Report targets found        If Succeed/Fail = 9/ 10
 9      Commence task on target     If Succeed/Fail = 10/ 9
 10     Transit to recovery point   If Succeed/Fail = 99/ 10
 11     Abort Mission(98)           If Succeed/Fail = 98/ 98
 12     Mission Complete(99)        If Succeed/Fail = 99/ 99
```

Figure 7. File as presented on screen during simulation run-time

I. SUMMARY

The current implementation of the RBM Strategic and Tactical Levels can be enhanced in several ways by a change in the implementing programming language. The use of object-oriented programming allows the simple replacement of subordinate objects when specialized behavior is needed for a particular mission. Both the LISP and Ada programming languages enable this mission flexibility where the current C programming language implementation does not. By the introduction of concurrent behavior among the Tactical Level objects representing the watch officers and their subordinates in the manned submarine paradigm, the entire mission can be performed more efficiently and without the need for complex error-handling routines required by the sequential cyclical executive approach. This behavior can be introduced using the Ada programming language with an object-oriented design model.

V. TESTING AND RESULTS

A. INTRODUCTION

Testing of the Ada versions of the Strategic and Tactical Level was accomplished using a SparcStation10 Sun workstation running Unix. The workstation environment was typical of the Sun workstations available in the department laboratories.

B. SCENARIO

The mission scenario chosen for development corresponds to the sample mission shown in Figure 3 of Chapter IV. The mission follows the same scenario to enable reasonable comparisons between implementations in different programming languages -- PROLOG and C, LISP alone, and Ada alone. The mission scenario follows this timeline:

- Vehicle initialization
- Vehicle launch
- Transit to first search location using waypoint-following
- Search at first location
- If targets found in search area, home to standoff distance and drop package
- Transit to second search location using waypoint-following
- Search at second location
- If targets found in search area, home to standoff distance and drop package
- Transit to recovery point using waypoint-following
- Vehicle Recovery

1. LISP Testing

LISP program files were developed following the LISP design model shown in Figure 4 of Chapter IV. Interactive test runs simulating both successful and aborted missions were executed. Traces of sample test runs are included in Appendix E.

2. Ada Testing

Ada program files were generated following the Ada design model shown in Figure 5 of Chapter IV. Interactive test runs simulating both successful and aborted missions were executed. Traces of sample test runs are included in Appendix E.

C. RESULTS

Both the LISP and Ada program versions successfully executed the test scenario without crashing and were able to appropriately handle multiple simulated AUV system failures during execution. The simulation testing was of an interactive nature, with the operator of the program providing responses from the Execution Level and from portions of the Tactical Level that were not implemented in this thesis. This interactive nature precluded quantitative comparison of the speed of execution of the program. Qualitative comparison of the implementations suggests that the Ada responses are somewhat faster.

D. SUMMARY

LISP and Ada program language versions of the Strategic and Tactical Levels of the Rational Behavior Model successfully execute the equivalent mission scenario of the existing combination of PROLOG and C code.

Implementation in an object-oriented language such as LISP or Ada provides the opportunity for significant code reuse and less code manipulation in the event that a change to a part of the code is desired, such as when a more accurate navigator is required for a specific mission.

The Ada version also simplifies the Strategic Level by specifying a mission file separate from the execution portion of the code. This adds the capability of dynamically changing the mission profile during the mission, such as when the Navigator must add a waypoint due to encountering a previously unknown obstacle.

VI. CONCLUSIONS AND FUTURE WORK

A. RESEARCH CONTRIBUTIONS

This thesis has demonstrated the effectiveness of the Ada programming language in providing reliable and maintainable software for the concurrent execution of tasks in autonomous vehicle control. In addition, within the framework of the Rational Behavior Model (RBM) Strategic and Tactical levels, it promotes the necessary abstraction at the Strategic Level so that the mission specialist needs only minimal knowledge of any programming language. Finally, it demonstrates that implementations of RBM in PROLOG and C can be efficiently programmed in Ada with concurrent execution, and compares favorably with non-concurrent programs in both PROLOG and LISP.

On a practical level, this thesis provides a strong start towards the software engineering needed for the fielding of robot minisubmarines as useful military (and non-military) systems. Such systems will present commanders with invaluable tools for reconnaissance, weapons, and defense.

For example, one of the major problems facing the Navy is the location and neutralization of near-shore shallow water mines. A major part of the solution is expected to be the use of autonomous vehicles (singly or in groups) that can detect, mark, and neutralize these threats to ships and amphibious landing craft. As a force multiplier, autonomous vehicles can also enhance the Navy's war-fighting capability while reducing operational costs in the reconnaissance role. To be effective, autonomous vehicles require a software architecture like RBM that readily supports rapid mission planning and reconfiguration concurrently with stable long-lived vehicle control.

B. SUGGESTIONS FOR FUTURE RESEARCH

This thesis forms a foundation for future work in advanced control software for autonomous vehicles within the framework of structured software engineering, especially as it pertains to combining the ongoing work of the NPS Computer Aided Prototyping System (CAPS) and the Autonomous Underwater Vehicle (AUV) Research Groups.

The CAPS group seeks to provide an efficient and effective means to capture the huge body of existing Ada software components into a reusable software base, and the AUV research group is developing the triple-layer Rational Behavior Model (RBM) software architecture for control of the NPS AUV *Phoenix* and other autonomous vehicles. Each group can benefit greatly from the work of the other. The AUV group can validate CAPS work on an existing military-use platform while the CAPS software base and tools can serve to consolidate and streamline the developing body of RBM components in a proven software engineering framework.

This thesis work has taken a significant first step in bringing the two together. The further conjunction of CAPS with RBM for control of autonomous vehicles is a strong candidate for future research. The development of a maintainable and adaptable suite of control software for autonomous vehicles, toward which this thesis represents a beginning, could save DoD both time and money in the development of future military systems both by demonstrating the validity and efficiency of using existing software components and improving structured autonomous vehicle control software.

Additional work is still required to develop the communications between Ada and C at the Tactical Level - Execution Level interface with the NPS AUV *Phoenix* and concurrently with the NPS Underwater Virtual World AUV Simulator. Once successful simulation in the Underwater Virtual World is satisfactory, this effort should lead directly to the in-water testing of this Ada version of the RBM on board the *Phoenix*.

Another extension of this work is the development of software tools for rapidly building executable mission files. Using either a menu-driven format or a graphical user interface, the mission specialist should be able to produce acceptable executable mission files with just a few keystrokes, either for immediate execution or for storage and later retrieval. For a system fielded to the fleet, this software "Mission Planning System" should typically be able to be operated by a mission specialist at the E-5 level; for example, a Fire

Controlman or Electronics Technician Second Class or equivalent. Such a system needs to be robust enough to avoid failures during mission planning sessions and should have built-in constraints on parameters so that mission files produced will be executable by the AUV within a defined acceptable range. For example, the Petty Officer performing mission planning should not be able to specify a mission file that includes depths, speeds, or ranges the AUV is not capable of achieving.

APPENDIX A. STRATEGIC LEVEL PROLOG CODE

This appendix contains the PROLOG code for the Strategic Level of the RBM. The Mission Controller was written by Dr. Robert McGhee and extended by the author. The remaining code was generated by the author as part of a class project for the CS4314 course in Symbolic Computing and was modelled on a more complex mission simulation produced by Ronald Byrnes, Dr. S. H. Kwak, Dr. Anthony Healey, and David Marco on Nov 12, 1992.

A. MISSION CONTROLLER

```
/*--------- Mission Controller below this line -----------------*/

affirmative(1).
affirmative(y).
affirmative(t).
not(X) :- \+ X.

go :- execute_mission.

execute_mission      :- initialize_mission, repeat, execute_phase, done.
  initialize_mission :- clean_up, asserta(current_phase(1)).
    clean_up :- abolish(current_phase,1),
                abolish(task_abort,1), abolish(search_abort,1),
                abolish(systems_abort,1),
                asserta(task_abort(0)), asserta(search_abort(0)),
                asserta(systems_abort(0)).

  execute_phase :- not(critical_systems_ok),
                   asserta(current_phase(mission_abort)).
    critical_systems_ok :- ask('Critical Systems OK',X), affirmative(X).
  execute_phase :- current_phase(X), execute_phase(X), !.

  done :- current_phase(mission_abort), command('Abort mission').
  done :- current_phase(mission_complete), command('Mission complete').
```

B. MISSION FILE

```
/*--------------- Mission File below this line ------------------*/

execute_phase(1)   :- command('Initialize vehicle'),
                             repeat, phase_completed(1).
phase_completed(1)  :- initialize_vehicle,
                       retract(current_phase(1)),
                       asserta(current_phase(2)).

execute_phase(2)    :- command('Transit to next waypoint'),
                       repeat, phase_completed(2).
phase_completed(2)  :- waypoint_control,
                       retract(current_phase(2)),
                       asserta(current_phase(21)).

execute_phase(21)   :- ask('Task location reached',X),
                       affirmative(X),
                       retract(current_phase(21)),
                       asserta(current_phase(3)).
execute_phase(21)   :- ask('Abort transit',X), affirmative(X),
                       retract(current_phase(21)),
                       asserta(current_phase(mission_abort)).
execute_phase(21)   :- retract(current_phase(21)),
                       asserta(current_phase(2)).


execute_phase(3)    :- command('Find sonar target'),
                       repeat, phase_completed(3).
phase_completed(3)  :- search,
                       retract(current_phase(3)),
                       asserta(current_phase(31)).

execute_phase(31)   :- ask('Target found',X), affirmative(X),
                       retract(current_phase(31)),
                       asserta(current_phase(4)).
execute_phase(31)   :- ask('Abort sonar search',X), affirmative(X),
                       retract(current_phase(31)),
                       asserta(current_phase(5)).
execute_phase(31)   :- retract(current_phase(31)),
                       asserta(current_phase(3)).


execute_phase(4)    :- command('Commence task on target'),
                       repeat, phase_completed(4).
phase_completed(4)  :- task,
                       retract(current_phase(4)),
                       asserta(current_phase(5)).
phase_completed(4)  :- ask('Abort target approach',X), affirmative(X),
                       retract(current_phase(4)),
                       asserta(current_phase(mission_abort)).


execute_phase(5)    :- command('Transit to next waypoint'),
                       repeat, phase_completed(5).
phase_completed(5)  :- waypoint_control,
                              retract(current_phase(5)),
                              asserta(current_phase(51)).

execute_phase(51)   :- ask('Task location reached',X),
                       affirmative(X),
                       retract(current_phase(51)),
                       asserta(current_phase(6)).
execute_phase(51)   :- ask('Abort transit',X), affirmative(X),
                       retract(current_phase(5)),
                       asserta(current_phase(mission_abort)).
execute_phase(51)   :- retract(current_phase(51)),
                       asserta(current_phase(5)).


execute_phase(6)    :- command('Find sonar target'),
                       repeat, phase_completed(6).
phase_completed(6) :- search,
                       retract(current_phase(6)),
                       asserta(current_phase(61)).

execute_phase(61)   :- ask('Target found',X), affirmative(X),
                       retract(current_phase(61)),
                       asserta(current_phase(7)).
execute_phase(61)   :- ask('Abort sonar search',X), affirmative(X),
                       retract(current_phase(6)),
                       asserta(current_phase(8)).
execute_phase(61)   :- retract(current_phase(61)),
                       asserta(current_phase(6)).


execute_phase(7)    :- command('Commence task on target'),
                       repeat, phase_completed(7).
phase_completed(7) :- task,
                       retract(current_phase(7)),
                       asserta(current_phase(8)).
phase_completed(7) :- ask('Abort target approach',X), affirmative(X),
                       retract(current_phase(7)),
                       asserta(current_phase(mission_abort)).


execute_phase(8)    :- command('Transit to recovery point'),
                       repeat, phase_completed(8).
phase_completed(8) :- waypoint_control,
                       retract(current_phase(8)),
                       asserta(current_phase(81)).

execute_phase(81)   :- ask('At recovery point',X),
                       affirmative(X),
                       retract(current_phase(81)),
                       asserta(current_phase(mission_complete)).
execute_phase(81)   :- ask('Abort transit',X), affirmative(X),
                      retract(current_phase(8)),
                      asserta(current_phase(mission_abort)).
execute_phase(81)  :- retract(current_phase(81)),
                      asserta(current_phase(8)).
```

C. MISSION DOCTRINE

```
/*---------Mission Doctrine below this line-----------------*/

initialize_vehicle :- repeat, initialization_completed.
  initialization_completed :- ask('Initialization completed',X),
                              affirmative(X).
  initialization_completed :- ask('Initialization aborted',X),
                              affirmative(X),
                              asserta(current_phase(mission_abort)).

waypoint_control :- get_waypoint_status, plan, send_setpoints_and_modes.
  get_waypoint_status :- reach_waypoint_p, get_next_waypoint, gps_check.

    gps_check :- systems_abort(1).
    gps_check :- gps_needed_p, get_gps_fix.
      gps_needed_p :-  ask('GPS fix needed',X), affirmative(X).
      get_gps_fix :- task_abort(1).
      get_gps_fix :- command('Get GPS fix'), repeat, gps_fix_complete.
        gps_fix_complete :- systems_abort(1), ask('Continue Mission',X),
                            affirmative(X), retract(systems_abort(1)),
                            asserta(systems_abort(0)).
        gps_fix_complete :- systems_abort(1),
                            asserta(current_phase(mission_abort)).
        gps_fix_complete :- ask('GPS fix obtained',X), affirmative(X).
        gps_fix_complete :- ask('Abort GPS fix',X), affirmative(X),
                            retract(systems_abort(0)),
                            asserta(systems_abort(1)), !, fail.
    gps_check.
    reach_waypoint_p  :- systems_abort(1).
    reach_waypoint_p  :- ask('Waypoint reached',X), affirmative(X).
    get_next_waypoint :- systems_abort(1).
    get_next_waypoint :- command('Get next waypoint'), repeat,
                         waypoint_obtained.
      waypoint_obtained :- systems_abort(1), ask('Continue Mission',X),
                           affirmative(X), retract(systems_abort(1)),
                           asserta(systems_abort(0)).
      waypoint_obtained :- systems_abort(1),
                           asserta(current_phase(mission_abort)).
      waypoint_obtained :- ask('Got next waypoint',X), affirmative(X).
      waypoint_obtained :- ask('Waypoint process abort',X),
affirmative(X),
                           retract(systems_abort(0)),
                           asserta(systems_abort(1)), !, fail.
  get_waypoint_status.

  plan :- systems_abort(1).
  plan :- not(noncritical_systems_ok ), global_replan.
    noncritical_systems_ok :- ask('NonCritical Systems OK',X),
                                  affirmative(X).
    global_replan :- command('Loiter'), command('Start global
replanner').
  plan :- near_uncharted_obstacle, local_replan.
    near_uncharted_obstacle :- unk_obstacle_p, log_new_obstacle.
      unk_obstacle_p :- ask('Area clear of uncharted obstacles',X),
                        not(affirmative(X)).
      log_new_obstacle :- command('Log new obstacle'),
                          repeat, new_obstacle_logged.
        new_obstacle_logged :- systems_abort(1), ask('Continue
Mission',X),
                               affirmative(X),
retract(systems_abort(1)),
                               asserta(systems_abort(0)).
        new_obstacle_logged :- systems_abort(1),
                               asserta(current_phase(mission_abort)).
        new_obstacle_logged :- ask('New obstacle logged',X),
                               affirmative(X).
        new_obstacle_logged :- ask('Log system failure',X),
                               affirmative(X),
retract(systems_abort(0)),
                               asserta(systems_abort(1)),
                               !, fail.
    local_replan  :-  systems_abort(1).
    local_replan  :- command('Loiter'), command('Start local
replanner').
  plan.

  send_setpoints_and_modes :- systems_abort(1).
  send_setpoints_and_modes :- command('Send setpoints and modes'),
repeat,
                              setpoints_and_modes_sent.
    setpoints_and_modes_sent :- systems_abort(1), ask('Continue
Mission',X),
                                affirmative(X),
retract(systems_abort(1)),
                                asserta(systems_abort(0)).
    setpoints_and_modes_sent :- systems_abort(1),
                                asserta(current_phase(mission_abort)).
    setpoints_and_modes_sent :- ask('Setpoints and modes sent',X),
                                affirmative(X).
    setpoints_and_modes_sent :- ask('Setpoints and modes system OK',X),
                                not(affirmative(X)),
                                retract(systems_abort(0)),
                                asserta(systems_abort(1)),
                                !, fail.
```

```
search :-  repeat, search_completed.
    search_completed :- search_abort(1), ask('Continue Mission',X),
                        affirmative(X), retract(search_abort(1)),
                        asserta(search_abort(0)).
    search_completed :- search_abort(1),
                        asserta(current_phase(mission_abort)).
    search_completed :- ask('Search pattern completed',X),
affirmative(X).
    search_completed :- ask('Sonar failure',X), affirmative(X),
                        retract(search_abort(0)),
asserta(search_abort(1)),
                        !, fail.


task :- homing, drop_package, get_gps_fix.
  homing :- repeat, homing_completed.
    homing_completed :- task_abort(1), ask('Continue Mission',X),
                        affirmative(X), retract(task_abort(1)),
                        asserta(task_abort(0)).
    homing_completed :- task_abort(1),
                        asserta(current_phase(mission_abort)).
    homing_completed :- ask('Standoff distance reached',X),
affirmative(X).
    homing_completed :- ask('Abort homing',X), affirmative(X),
                        retract(task_abort(0)), asserta(task_abort(1)),
                        !, fail.

  drop_package :- task_abort(1).
  drop_package :- command('Drop package'), repeat,
package_drop_complete.
    package_drop_complete :- task_abort(1), ask('Continue Mission',X),
                             affirmative(X), retract(task_abort(1)),
                             asserta(task_abort(0)).
    package_drop_complete :- task_abort(1),
                             asserta(current_phase(mission_abort)).
    package_drop_complete :- ask('Is package dropped',X),
affirmative(X).
    package_drop_complete :- ask('Is package drop aborted',X),
                             affirmative(X), retract(task_abort(0)),
                             asserta(task_abort(1)),
                             !, fail.
```

D. SIMULATED TACTICAL LEVEL

```
   /*------Simulated Tactical Level below this line ----------------*/


ask(Q,A) :- write(Q), write('?'), nl, read(A), nl.
command(X) :- write(X), write('!'), nl.
```

APPENDIX B. TACTICAL LEVEL C CODE

This appendix contains the C code for the currently implemented Tactical Level of the RBM. The code was generated by Dr. Anthony Healey and David Marco as part of a the ongoing AUV project and represents the current state of the code as of September 1995.

A. FILE "tactical.c"

```
#include "tactical.h"

  int socket_descriptor,socket_accepted,socket_stream;
  int socket_length = 81;
  int bytes_received, bytes_read, bytes_written,
                          bytes_left, bytes_sent;
  int iris_socket_descriptor,iris_socket_accepted,iris_socket_stream;
  char remote_host_name[60];
  int     shutdown_signal_received;

  time_t time1,time2;  /* For Time Out Functions */

  int DEPTH_FILTER_ON;

  int n_setpts,current_setpt_index;
  int set_pt_code[10];
  double time_out[10];
  double X_setpt[10],Y_setpt[10],z_setpt[10];
  double sigma_xf_min[10],sigma_yf_min[10],sigma_zf_min[10];
  double sigma_phif_min[10],sigma_thetaf_min[10],sigma_psif_min[10];
  double phi_setpt[10],theta_setpt[10],psi_setpt[10];
  int st1000_sweep_mode[10];       /* 1 = continuous sweep  */
                          /* 2 = directional sweep */
  double st1000_scan_direction[10];
  double st1000_sweep_width[10];
  int st1000_nsteps[10];

  double X_com,Y_com,Z_com;
  double phi_com,theta_com,psi_com;

  double x,y,z,z_dot,u,v,w;
  double phi,theta,psi,p,q,r;
  double psi_st725,psi_st1000;

  int INT_CONTROL_Z;

  double t;

  int n_sonar_cycles = 0;

  int IRIS_CONNECTED = 0;

  int GYROS_ON = 0;

  FILE *mission_infp;

  FILE *x_error_outfp,*z_error_outfp,*psi_error_outfp;

  int st1000_direction = 1;
  int psi_st1000_psi_sonar_count = 0;
  int st1000_nsteps_count = 0;

  int print_depth_info_count = 11;

  double sonar_target_avg;

/* Shutdown Function */
void shutdown_os9sender ()
{
  shutdown_signal_received = TRUE;

  if (close (socket_stream) == -1)
     printf ("sunsender close (socket_stream) failed\n");
     printf("SUN and OS9 Dis-Connected\n");
  return;

}


/* Shutdown Conn. to Iris Function */
void shutdown_sun_to_iris()
{
  char command_sent[81];
  shutdown_signal_received = TRUE;
  /* Tell Iris To Jump Out of Telemetry Loop and Terminate Normally */
  sprintf(command_sent,"%s","shutdown");
  write_to_iris(command_sent);

  sleep(1);

  if (close (iris_socket_stream) == -1)
     printf ("os9server close (iris_socket_stream) failed\n");
     printf("SUN and IRIS Dis-Connected\n");
  return;
}


/* NONBLOCKING Socket Modification Function for SUN!!!*/

dontblock(spath)
{
  if(fcntl(spath, F_SETFL,FNDELAY) < 0)
  {
    perror("fcntl F_SETFL, FNDELAY");
    exit(1);
  }
}


initialize_sunsender()
{
  struct sockaddr_in server_address;
  char             test_buffer [81];
  static char       *ptr;
  register struct hostent *server_entity;

  /* start by finding default/desired remote host to connect to */
  {
    server_entity = gethostbyname ("auv");
    if (server_entity == NULL)
    {
      printf("os9sender -remote host (\"%s\") not found\n",
            remote_host_name);
      exit (-1);
    }
  /* Client opens server port */

  /* Fill in structure 'server_address' with the address of the */
  /*        remote host (i.e. SERVER) that we want to connect with: */

  server_address.sin_family   = AF_INET;

  /* copy server IP address into sockaddr_in struct server_address */
  strncpy(&(server_address.sin_addr.s_addr), server_entity->h_addr,
        server_entity->h_length);

  /* make sure port is in network byte order */
  server_address.sin_port = htons (AUVSIM1_TCP_PORT_0);

  /* Open TCP (Internet stream) socket */
  if ( (socket_descriptor = socket (AF_INET, SOCK_STREAM, 0)) < 0 )
  {
    printf ("os9sender client can't open server stream socket");
    exit (-1);
  }

  /* Connect to the server.  Process will block/sleep until connection is
       is established.  Timeout will return an error. */
  if (connect (                socket_descriptor,
             (struct sockaddr *) &server_address,
                        sizeof (server_address)) < 0)
  {
    printf ("os9sender client can't connect to server socket\n");
    exit (-1);
  }

} /* end initialization */

/* loop transferring telemetry until shutdown_signal_received: */

/* Two-way reflector:  listen to local program and relay to remote host,
                  listen to  remote host  and relay to local program */

socket_stream = socket_descriptor;   /* client */

/* test handshakes */

write (socket_stream, "SUCCESS #2:  SUN connected to IRIS!", 46);
```

```
  read  (socket_stream, test_buffer, 46);
  test_buffer [47] = '\n';
  /*printf ("test handshake between hosts: \n%s\n", test_buffer);*/

  dontblock(socket_stream); /* Set Socket Up for NONBLOCKING, otherwise */
                    /* read function will hang if no data on buffer */
}


connect_sun_to_iris()
{
  struct sockaddr_in iris_server_address;
  char             test_buffer [81];
  static char      *ptr;

/* Initialize server */

  /* setup to listen for client to attempt connection */
  {
    /* Server opens server port to IRIS */

    /* Open TCP (Internet stream) in socket */
    if ( (iris_socket_descriptor = socket (AF_INET, SOCK_STREAM, 0)) < 0 )
    {
      printf ("os9server can't 'open' IRIS stream socket");
      exit (-1);
    }

    /* Bind local address so client can talk to server */
    iris_server_address.sin_family = AF_INET;  /* Internet protocol family */

    /* make sure port is in network byte order IRIS */
    iris_server_address.sin_addr.s_addr = htonl (INADDR_ANY);
    iris_server_address.sin_port       = htons (AUVSIM1_TCP_PORT_2);

    if (bind (              iris_socket_descriptor,
          (struct sockaddr *) &iris_server_address,
                  sizeof (iris_server_address)) < 0)
    {
      printf("os9server socket 'bind' unsuccessful IRIS\n");
      if(errno == 0x70d)
      {
        printf(" errno = EADDRINUSE, address already in use IRIS\n");
```

```
            }
            exit (-1);
          }

          /* prepare socket queue for connection requests using listen IRIS */
          /* Server 'accept' waits for client connections IRIS */

          bytes_received = sizeof (iris_socket_descriptor);

          while ((iris_socket_accepted  = accept ( iris_socket_descriptor,
                                  &iris_server_address,
                                  &bytes_received)) < 1)     /* block */



          printf("os9server connection is open between networks IRIS.\n");
          printf("os9server connection is open between networks IRIS.\n");


        } /* end initialization */

        /* loop transferring telemetry until shutdown_signal_received: */

        /* Two-way reflector:  listen to local program and relay to remote host,
                        listen to  remote host  and relay to local program */

        iris_socket_stream = iris_socket_accepted;   /* server */

        /* test handshakes */

        write(iris_socket_stream,"SUCCESS #1: SUN connected to IRIS!",
                        46);
        read  (iris_socket_stream, test_buffer, 46);
        test_buffer [47] = '\n';
        printf ("test handshake between hosts IRIS: \n%s\n", test_buffer);

        dontblock(iris_socket_stream); /* Set Socket Up for NONBLOCKING, otherwise */
   /* read function will hang if no data on buffer */
       }


      write_to_os9(command_sent)
```

```
  char command_sent[];
{
  static char *ptr;

  /* read from local stdin, relay to remote host */

  bytes_received = strlen (command_sent);

  if(bytes_received < 0)  /* read failure */
  {
    printf ("os9sender gets () from keyboard unsuccessful\n");
    shutdown_os9sender();
  }

  if (bytes_received > socket_length)

    listen (iris_socket_descriptor, SOCKET_QUEUE_SIZE);

  {
    printf ("os9sender send_telemetry_to_server error: ");
    printf ("bytes_received too big for packet socket_length\n");
    printf ("        ");
    printf ("[bytes_received=%d] > [socket_length=%d]; ",
              bytes_received,      socket_length);
    printf ("string truncated\n");
  }

  bytes_left      = socket_length;
  bytes_written    = 0;
  ptr            = command_sent;

  while ((bytes_left > 0) && (bytes_written >= 0))  /* write loop */
  {
    bytes_sent = write (socket_stream, ptr, bytes_left);

    if     (bytes_sent <  0) bytes_written = bytes_sent;
    else if (bytes_sent >  0)
    {
      bytes_left    -= bytes_sent;
      bytes_written  += bytes_sent;
      ptr          += bytes_sent;
    }
  }
  if(bytes_written < 0)
  {
    printf ("os9sender send_telemetry_to_server () send failed, ");
    printf ("%d bytes_written\n", bytes_written);
  }
}


write_to_iris(command_sent)
  char command_sent[];
{
  static char *ptr;

  /* read from local stdin, relay to remote host */

  bytes_received = strlen (command_sent);

  if (bytes_received > socket_length)
  {
    printf ("os9sender send_telemetry_to_server error: ");
    printf ("bytes_received too big for packet socket_length\n");
    printf ("          ");
    printf ("[bytes_received=%d] > [socket_length=%d]; ",
              bytes_received,     socket_length);
    printf ("string truncated\n");
  }

   bytes_left      = socket_length;
  bytes_written    = 0;
  ptr              = command_sent;

  while ((bytes_left > 0) && (bytes_written >= 0))  /* write loop */
  {
    bytes_sent = write (iris_socket_stream, ptr, bytes_left);

    if      (bytes_sent <  0) bytes_written = bytes_sent;
    else if (bytes_sent >  0)
    {
      bytes_left    -= bytes_sent;
      bytes_written  += bytes_sent;
      ptr            += bytes_sent;
    }
  }

  if(bytes_written < 0)
  {
    printf ("os9sender send_telemetry_to_server () send failed, ");
    printf ("%d bytes_written\n", bytes_written);
  }
}


/* Will Loop Until Something to Read */
read_from_os9(command_read)
  char *command_read;
{
  char command_received[81];
  static char       *ptr;

  bytes_left    = socket_length;
  bytes_received = 0;
  ptr           = command_received;

  bytes_read = -1; /* Assume Not There Yet */
  while(bytes_read < 0)
  {
    bytes_read = read(socket_stream, ptr, bytes_left);
  }

  strcpy(command_read,command_received);
}


/* If Nothing To Read Will Exit Function and Return < 0 */
int read_from_iris(command_read)
  char *command_read;
{
  char command_received [81];
  static char       *ptr;

  /* listen to remote host, relay to local network/program */

  bytes_left       = socket_length;
  bytes_received    = 0;
  ptr              = command_received;
  bytes_read = read (iris_socket_stream, ptr, bytes_left);
  if(bytes_read > 0)
  {
    strcpy(command_read,command_received);
  }
  return(bytes_read);
}


int ood(command)
  char command[];
{
  int i,network_status,status,cage_indicate;
  int st725_init_status,st1000_init_status;
  char command_sent[81],command_read[81];

  if(strcmp(command,"start_sun_network") == 0)
  {
    initialize_sunsender();
    return(1);
  }

  else if(strcmp(command,"start_sun_and_iris_network") == 0)
  {
    initialize_sunsender();
    printf("Waiting to Connect with IRIS...\n");
    connect_sun_to_iris();
    IRIS_CONNECTED = TRUE;
    return(1);
  }

  else if(strcmp(command,"start_networks") == 0)
  {
    initialize_sunsender();
    printf("Waiting for Network Up Signal from Execution Level...\n");
    read_from_os9(&command_read[0]);
    sscanf(command_read,"%d",&network_status);
    printf("Network Up!\n");
    return(network_status);
  }

  else if(strcmp(command,"initialize_boards") == 0)
```

```
{
  write_to_os9("INITIALIZE_BOARDS");
  return(TRUE);
}

else if(strcmp(command,"turn_on_prop_power") == 0)
{
  write_to_os9("TURN_ON_PROP_POWER");
  return(TRUE);
}

else if(strcmp(command,"turn_off_prop_power") == 0)
{
  write_to_os9("TURN_OFF_PROP_POWER");
  return(TRUE);
}

else if(strcmp(command,"turn_on_sonar_power") == 0)
{
  write_to_os9("TURN_ON_SONAR_POWER");
  return(TRUE);
}

else if(strcmp(command,"turn_off_sonar_power") == 0)
{
  write_to_os9("TURN_OFF_SONAR_POWER");
  return(TRUE);
}

else if(strcmp(command,"gyros_on") == 0)
{
  GYROS_ON = TRUE;
  return(TRUE);
}

else if(strcmp(command,"zero_sensors") == 0)
{
  if(GYROS_ON)
  {
    write_to_os9("ZERO_GYROS_AND_DEPTH_CELL");
  }
  else
  {
      write_to_os9("ZERO_DEPTH_CELL");
    }
    return(TRUE);
  }

  else if(strcmp(command,"initialize_st1000_sonar") == 0)
  {
    write_to_os9("INITIALIZE_ST1000_SONAR");

    read_from_os9(&command_read[0]);
    sscanf(command_read,"%d",&st1000_init_status);
    return(st1000_init_status);
  }

  else if(strcmp(command,"initialize_st725_sonar") == 0)
  {
    write_to_os9("INITIALIZE_ST725_SONAR");
    read_from_os9(&command_read[0]);
    sscanf(command_read,"%d",&st725_init_status);
    return(st725_init_status);
  }

  else if(strcmp(command,"uncage_directional_gyroscope") == 0)
  {
    write_to_os9("UNCAGE_DIRECTIONAL_GYROSCOPE");
    read_from_os9(&command_read[0]);
    sscanf(command_read,"%d",&cage_indicate);
    return(!cage_indicate);
  }

  else if(strcmp(command,"initialization_done") == 0)
  {
    write_to_os9("INITIALIZATION_DONE");
    return(TRUE);
  }

  else if(strcmp(command,"shutdown_network") == 0)
  {
    write_to_os9("shutdown");
    shutdown_os9sender();

/*      if(IRIS_CONNECTED) shutdown_sun_to_iris();*/
```

```
        /*close(x_error_outfp);
        close(z_error_outfp);
        close(psi_error_outfp);*/

        return(1);
      }
      else if(strcmp(command,"read_mission_file") == 0)
      {
        if((mission_infp = fopen("mission.d","r")) == 0)  /* Open file for
                                                              reading */
        {
          printf("Cannot Open File mission.d\n");
          return(0);
        }
        else
        {

          /* Nominals for sigma_xf_min = 0.3;
                        sigma_yf_min = 0.0; N/A
                        sigma_zf_min = 0.1;
                        sigma_phif_min = 0.0; N/A
                        sigma_thetaf_min  = 0.0; N/A
                        sigma_psif_min = 0.1;
          */

          /* st1000_sweep_mode = 1 for Cont sweep CW, -1 for CCW, */
          /* 2 for sector Sweep */
          fscanf(mission_infp,"%d",&n_setpts);
          printf("%d\n",n_setpts);
          for(i=0;i<n_setpts;++i)
          {
            fscanf(mission_infp,"%F %F %F %F %F %F %F %F %F %F %F %F %F %d
%F %F",
                  &time_out[i],
                  &X_setpt[i],
                  &Y_setpt[i],
                  &z_setpt[i],
                  &phi_setpt[i],
                  &theta_setpt[i],
                  &psi_setpt[i],
                  &sigma_xf_min[i],
                  &sigma_yf_min[i],
                  &sigma_zf_min[i],
            &sigma_phif_min[i],
            &sigma_thetaf_min[i],
            &sigma_psif_min[i],
            &st1000_sweep_mode[i],
            &st1000_scan_direction[i],
            &st1000_sweep_width[i]);
      printf("%f %f %f %f %f %f %f %f %f %f %f %f %f %d %f %f\n",
            time_out[i],
            X_setpt[i],
            Y_setpt[i],
            z_setpt[i],
            phi_setpt[i],
            theta_setpt[i],
            psi_setpt[i],
            sigma_xf_min[i],
            sigma_yf_min[i],
            sigma_zf_min[i],
            sigma_phif_min[i],
            sigma_thetaf_min[i],
            sigma_psif_min[i],
            st1000_sweep_mode[i],
            st1000_scan_direction[i],
            st1000_sweep_width[i]);
    }

    fclose(mission_infp);
    current_setpt_index = -1; /* Have not encounted first setpt */
    DEPTH_FILTER_ON = FALSE;  /* Not on During Init */

    /* Open Pos/Vel Error Files for Writing */
    x_error_outfp   = fopen("x_error.d","w");
    z_error_outfp   = fopen("z_error.d","w");
    psi_error_outfp = fopen("psi_error.d","w");

    printf("File opened successfully\n");
    return(1); /* File opened successfully */
  }
}

else
{
  printf("Command %s Not Recognized for predicate ood!\n",command);
  return(0);
  }
}


int engineer(command)
  char command[];
{
  return(1);
}


int exec_sleep(time)
  int time; /* Time to Sleep in Seconds */
{
  sleep(time);
  return(1);
}

int exec_start_timer()
{
  time_t t_null;

  time1 = time(&t_null);
  return(1);
}

int ask_time_out()
{
  double delta_time;
  time_t t_null;

  time2 = time(&t_null);
  delta_time = (float) (time2-time1);

  if(delta_time > time_out[current_setpt_index])
  {
    return(1);
  }
  else
  {
    return(0);
  }
```

```
}


int ask_system_problem()
{
  return(0); /* No system Problem */
}

int exec_next_setpt_data()
{
  current_setpt_index = current_setpt_index + 1;
  printf("current_setpt_index = %d\n",current_setpt_index);
  return(TRUE);
}

int ask_int_control_z_on()
{
  if(INT_CONTROL_Z)
  {
    return(TRUE);
  }
  else
  {
    return(FALSE);
  }
}

int ask_int_control_z_off()
{
  if(INT_CONTROL_Z)
  {
    return(FALSE);
  }
  else
  {
    return(TRUE);
  }
}

int exec_start_int_control_z()
{
  char command_sent[81],command_read[81];
  INT_CONTROL_Z = TRUE;

  sprintf(command_sent,"%s","START_INT_CONTROL_Z");
  write_to_os9(command_sent);
  return(TRUE);
}

int ask_depth_filter_off()
{
  if(DEPTH_FILTER_ON)
  {
    printf("Inside ask_depth_filter_off Returning FALSE\n");
    return(FALSE);
  }
  else
  {
    printf("Inside ask_depth_filter_off Returning TRUE\n");
    return(TRUE);
  }
}


int exec_start_depth_filter()
{
  char command_sent[81],command_read[81];

  DEPTH_FILTER_ON = TRUE;

  sprintf(command_sent,"%s","START_DEPTH_FILTER");
  write_to_os9(command_sent);
  return(TRUE);
}

int exec_start_depth_error_filter()
{
  char command_sent[81],command_read[81];

  sprintf(command_sent,"%s","START_DEPTH_ERROR_FILTER");
  write_to_os9(command_sent);
  return(TRUE);
}

int exec_start_heading_error_filter()
```

```
{
  char command_sent[81],command_read[81];

  sprintf(command_sent,"%s","START_HEADING_ERROR_FILTER");
  write_to_os9(command_sent);
  return(TRUE);
}

int exec_start_X_error_filter()
{
  char command_sent[81],command_read[81];

  sprintf(command_sent,"%s","START_X_ERROR_FILTER");
  write_to_os9(command_sent);
  return(TRUE);
}


int exec_submerge()
{
  char command_sent[81],command_read[81];

  sprintf(command_sent,"%s %f %f","SUBMERGE",z_setpt[current_setpt_index],
                          theta_setpt[current_setpt_index]);
  write_to_os9(command_sent);
  printf("z_setpt = %f theta_setpt = %f\n",z_setpt[current_setpt_index],
       theta_setpt[current_setpt_index]);
  return(1);
}

int exec_rotate()
{
  char command_sent[81],command_read[81];

  sprintf(command_sent,"%s %f","ROTATE",psi_setpt[current_setpt_index]);
  write_to_os9(command_sent);
  return(1);
}

int exec_surface()
{
  char command_sent[81],command_read[81];
  sprintf(command_sent,”%s”,”SURFACE”);
  write_to_os9(command_sent);
  return(1);
}

int ask_depth_reached()
{
  char command_sent[81],command_read[81];
  double z_est,sigma_zf;

  sprintf(command_sent,”%s”,”GET_DEPTH_INFO”);
  write_to_os9(command_sent);
  read_from_os9(&command_read[0]);
  sscanf(command_read,”%F %F”,&z_est,&sigma_zf);

  printf(“%f %f\n”,z_est,sigma_zf);
  /*fprintf(z_error_outfp,”%f %f\n”,z_est,sigma_zf); */

  if( sigma_zf < sigma_zf_min[current_setpt_index] )
  {
    printf(“Depth @ switch = %f\n”,z_est);
    printf(“sigma_zf @ switch = %f\n”,sigma_zf);
    return(TRUE);
  }
  else
  {
    return(FALSE);
  }
}

int ask_heading_reached()
{
  char command_sent[81],command_read[81];

  double sigma_psif;

  sprintf(command_sent,”%s”,”GET_HEADING_INFO”);
  write_to_os9(command_sent);
  read_from_os9(&command_read[0]);
  sscanf(command_read,”%F”,&sigma_psif);

  /* fprintf(psi_error_outfp,”%f\n”,sigma_psif);*/
  printf("%f\n",sigma_psif);
  if( sigma_psif < sigma_psif_min[current_setpt_index] )
  {
    printf("sigma_psif @ switch = %f\n",sigma_psif);

    return(TRUE);
  }
  else
  {
    return(FALSE);
  }
}


int ask_X_reached()
{
  char command_sent[81],command_read[81];
  double sigma_xf;

  sprintf(command_sent,"%s","GET_SERVO_X_DATA");
  write_to_os9(command_sent);
  read_from_os9(&command_read[0]);
  sscanf(command_read,"%F",&sigma_xf);

  fprintf(x_error_outfp,"%f\n",sigma_xf);

  if( sigma_xf < sigma_xf_min[current_setpt_index] ) /* Was 0.3 */
  {
    printf("sigma_xf @ switch = %f\n",sigma_xf);
    return(TRUE);
  }
  else
  {
    return(FALSE);
  }
}


int ask_surface_reached()
{
  char command_sent[81],command_read[81];

  double z_est,sigma_zf;
```

```
 sprintf(command_sent,"%s","GET_DEPTH_INFO");
 write_to_os9(command_sent);
 read_from_os9(&command_read[0]);
 sscanf(command_read,"%F %F",&z_est,&sigma_zf);

 if(fabs(0.0-z_est) <= 0.2)
 {
   return(TRUE);
 }
 else
 {
   return(FALSE);
 }
}

int exec_set_st1000_mode()
{
  char command_sent[81],command_read[81];

  sprintf(command_sent,"%s %d %f %f","SET_ST1000_MODE",
                 st1000_sweep_mode[current_setpt_index],
                 st1000_scan_direction[current_setpt_index],
                 st1000_sweep_width[current_setpt_index]);
  write_to_os9(command_sent);
  return(TRUE);
}


int exec_stop_ping_st1000_sonar()
{
  char command_sent[81],command_read[81];

  sprintf(command_sent,"%s","STOP_PING_ST1000");
  write_to_os9(command_sent);
  return(1);
}


int exec_start_send_st1000_data()
{
  char command_sent[81],command_read[81];
  sprintf(command_sent,"%s","START_SEND_ST1000_DATA");
  write_to_os9(command_sent);
  return(TRUE);
}


int exec_stop_send_st1000_data()
{   char command_sent[81],command_read[81];

  sprintf(command_sent,"%s","STOP_SEND_ST1000_DATA");
  write_to_os9(command_sent);
  return(TRUE);
}


int exec_find_sonar_target()
{
  int i,n;
  char command_sent[81],command_read[81];
  double avg,delta_range,range_vector[10];

  delta_range = 0.1;
  n = 5;

  /* Ping ST1000 n Consecutive Times, One Ping per Time Step */

  avg = 0.0;

  sprintf(command_sent,"%s %c","PING_ST1000",'Z');
  write_to_os9(command_sent);

  for(i=0;i<n;++i)
  {
    sprintf(command_sent,"%s","GET_ST1000_DATA");
    write_to_os9(command_sent);
    read_from_os9(&command_read[0]);
    sscanf(command_read,"%F %F",&psi_st1000,&range_vector[i]);
    printf("range_vector[%d] = %f\n",i,range_vector[i]);
    avg = avg + range_vector[i];
    printf("avg = %f @ i = %d\n",avg,i);
  }

  printf("float n = %f\n",((float) n));
  avg = avg/((float) n);
  sonar_target_avg = avg;

  printf(" \n");
  printf("************ => sonar_target_avg = %f\n",sonar_target_avg);
  printf(" \n");

  /* Prefilter zero returns or "too short" returns */
  if(avg < 1.0)
  {
    n_sonar_cycles = n_sonar_cycles + 1;
    return(FALSE);
  }

  /* Median Filter */
  for(i=0;i<n;++i)
  {
    if( (range_vector[i] < avg+delta_range) &&

        (range_vector[i] > avg-delta_range) )
    {
      /* Range Near Mean */
    }
    else
    {
      /* InConsistent Ranges */
      /* Increment n_sonar_cycles */
      n_sonar_cycles = n_sonar_cycles + 1;
      return(FALSE);
    }
  }

  /* All n Ranges Within delta */
  /* Reset n_sonar_cycles */
  printf("GOOD RANGES: avg = %f\n",avg);
  n_sonar_cycles = 0;

  return(TRUE);
}


int ask_sonar_ping_out()
{
  if(n_sonar_cycles > 10)
  {
    /* Have pinged for 10 Cycles and still not Consistent: Abort */
    return(TRUE);
  }
  else
  {
    return(FALSE);
  }
}

int ask_st1000_initialized()
{
  char command_sent[81],command_read[81];
  int init_status;

  sprintf(command_sent,"%s","GET_ST1000_INITIALIZATION_STATUS");
  write_to_os9(command_sent);

  read_from_os9(&command_read[0]);
  sscanf(command_read,"%d",&init_status);
  /*printf("init_status = %d\n",init_status);*/
  if(init_status == 1)
  {
    return(TRUE);
  }
  else if(init_status == 0)
  {
    return(FALSE);
  }
  else
  {
    printf("Don't Recognize ST1000_INITIALIZATION_STATUS\n");
    return(-1);
  }

}

int exec_start_sonar_filter()
{
  /* Start the Sonar Filter */
  char command_sent[81],command_read[81];
  printf("sonar_target_avg = %f\n",sonar_target_avg);

  sprintf(command_sent,"%s %d","START_SONAR_FILTER",sonar_target_avg);
  write_to_os9(command_sent);
  return(TRUE);
}

int exec_servo_X()
{
  char command_sent[81],command_read[81];

  sprintf(command_sent,"%s %f","SERVO_X",X_setpt[current_setpt_index]);
  write_to_os9(command_sent);
  return(TRUE);
}

int exec_stop_servo_X()
{
  char command_sent[81],command_read[81];

  sprintf(command_sent,"%s %f","STOP_SERVO_X");
  write_to_os9(command_sent);
  return(TRUE);
}
```

B. FILE "mission.d"

```
3
40.0  0.0 0.0 2.0 0.0 0.0 0.0 0.1 0.0 0.2 0.0 0.0 0.1 2 0.0 30.0
60.0  0.0 0.0 2.0 0.0 0.0 0.0 0.1 0.0 0.2 0.0 0.0 0.1 2 0.0 30.0
1.0   0.0 0.0 2.0 0.0 0.0 0.0 0.1 0.0 0.2 0.0 0.0 0.1 2 0.0 30.0
```

APPENDIX C. STRATEGIC AND TACTICAL LEVEL LISP CODE

This appendix contains the LISP code for the Strategic and Tactical Levels of the RBM. Lisp File “co.cl” contains the Mission Controller Block, Mission File Block, And Doctrine Blocks. Lisp File “ood.cl” contains the subordinate officer classes that implement the doctrines. For clarity, they are broken out as shown below. This code was generated as part of a class project for the CS4314 course in Symbolic Computing and involved the efforts of the author, Eric Bachmann, Michael Burns, Michael Campbell, David Gay and Brad Leonhardt.

A. MISSION CONTROLLER BLOCK

```
;contained in the File "co.cl"

(defun execute_mission ()
  (initialize_mission)
  (write-line " ")
  (do () ((done) 'done) (execute_phase *current_phase*)) )
```

B. MISSION FILE BLOCK

```
;contained in the File "co.cl"

;----------------- mission file block -----------------------
(defun issue_orders (phase)
  (case phase
    (1 (CO-command 'Initialize_vehicle))
    (2 (CO-command 'Transit_to_task_location))
    (3 (CO-command 'Search_area_for_target))
    (4 (CO-command 'Commence_task_on_target))
    (5 (CO-command 'Transit_to_task_location))
    (6 (CO-command 'Search_area_for_target))
    (7 (CO-command 'Commence_task_on_target))
    (8 (CO-command 'Transit_to_recovery_point))
    (t 1) ))

(defun next_phase (phase)
  (cond ((equal *current_phase* 'mission_abort) t)
        ((case phase
           (1  (if (= *complete* 1)  (setf *current_phase* 2)
                                                (setf *current_phase*
'mission_abort)))
```

```
          (2  (if (= *complete* 2)  (setf *current_phase* 3)))
          (3  (if (= *complete* 3)  (setf *current_phase* 31)))
          (31 (if (= *complete* 31) (setf *current_phase* 4)
                                    (setf *current_phase* 5)))
          (4  (if (= *complete* 4)  (setf *current_phase* 5)))
          (5  (if (= *complete* 5)  (setf *current_phase* 6)))
          (6  (if (= *complete* 6)  (setf *current_phase* 61)))
          (61 (if (= *complete* 61) (setf *current_phase* 7)
                                    (setf *current_phase* 8)))
          (7  (if (= *complete* 7)  (setf *current_phase* 8)))
                (8   (if  (= *complete* 8)   (setf  *current_phase*
'mission_complete)))
          (t  1) ))))

(defun phase_completed (phase)
  (case phase

    (1  (cond ((CO-ask 'Initialization_complete)
               (setf *complete* 1))
               ((CO-ask 'Initialization_aborted) t)
               (t nil) ))

    (2  (cond ((CO-ask 'Task_location_reached)
               (setf *complete* 2))
               (t (waypoint_control)) ))

    (3  (cond ((sonar_search) (setf *complete* 3))))
    (31 (cond ((CO-ask 'Target_found) (setf *complete* 31))
               (t 1) ))

    (4  (cond ((do_task)(setf *complete* 4))))

    (5  (cond ((CO-ask 'Task_location_reached)
               (setf *complete* 5))
               (t (waypoint_control)) ))

    (6  (cond ((sonar_search) (setf *complete* 6))))
    (61 (cond ((CO-ask 'Target_found) (setf *complete* 61))
               (t 1) ))

    (7  (cond ((do_task) (setf *complete* 7))))

    (8  (cond ((CO-ask 'At_recovery_point)
               (setf *complete* 8))
               (t (waypoint_control)) )) ))
```

C. DOCTRINE BLOCKS

```
;contained in the File "co.cl"

;---------------- general doctrine block ------------------

(setf ood1 (make-instance 'oodclass))

  (defun initialize_mission ()
    (setf *current_phase* 1  *complete* 0 )
    (setf *task_abort* 0  *search_abort* 0  *systems_abort* 0) )

  (defun done ()
    (cond ((equal *current_phase* 'mission_abort)
           (and (CO-command 'Abort_mission)
                (equal *current_phase* 'mission_abort) ))
          ((equal *current_phase* 'mission_complete)
           (CO-command 'Mission_complete) )
          (t nil)))

  (defun execute_phase (phase)
    (cond
      ((not (critical_systems_OK))
       (setf *current_phase* 'mission_abort))
      (t (and (issue_orders phase)
              (do ()
                ((phase_completed phase)
                 (next_phase phase) ))))))

      (defun critical_systems_OK ()
          (cond  ((CO-ask 'Critical_Systems_OK) t)
                 (t nil) ))

(defun CO-command (string)
  (write-line " ")
  (write *current_phase*) (write-string ".")
  (write-string "CO -->OOD: ")
  (write string)
  (write-line "!")
  (order ood1 string) )

(defun CO-ask (string)
  (write-line " ")
  (write *current_phase*) (write-string ".")
  (write-string "CO -->OOD: ")
  (write string) (write-line "?")
  (member (query ood1 string) *affirmative*) )

(setf *affirmative* '(1 t y yy))


;----------- waypoint_control doctrine block ----------------

(defun waypoint_control ()
  (and (get_waypoint_status)(plan)(send_setpoints_and_modes)) )
```

```
(defun get_waypoint_status ()
  (cond ((= *systems_abort* 1) t)
        ((reach_waypoint_p)
         (and (get_next_waypoint)(gps_check)))
        (t 1) ))

  (defun reach_waypoint_p ()
    (cond ((= *systems_abort* 1) t)
          ((CO-ask 'Waypoint_reached)
           (CO-command 'Get_next_waypoint))
          ((not (CO-ask 'Waypoint_process_OK))
           (and (setf *current_phase* 'mission_abort)
                (setf *systems_abort* 1) ))
          (t nil) ))

  (defun get_next_waypoint ()
    (cond
      ((= *systems_abort* 1) t)
      ((CO-ask 'Got_next_waypoint) t)
      ((not (CO-ask 'Waypoint_process_OK))
       (and (setf *current_phase* 'mission_abort)
            (setf *systems_abort* 1) ))
      (t (get_next_waypoint)) ))

  (defun gps_check ()
    (cond
      ((= *systems_abort* 1) t)
      ((CO-ask 'GPS_fix_needed)
       (and (CO-command 'Get_GPS_fix) (get_gps_fix)) )
      (t 1) ))

    (defun get_gps_fix ()
      (cond
        ((= *task_abort* 1) t)
        ((= *systems_abort* 1) t)
        ((CO-ask 'GPS_fix_obtained) t)
        ((CO-ask 'Abort_GPS_fix)
         (and (setf *current_phase* 'mission_abort)
              (setf *systems_abort* 1)))
        (t (get_gps_fix)) ))


(defun plan ()
  (cond
    ((= *systems_abort* 1) t)
    ((and (not (noncritical_systems_OK)) (global_replan)) t)
    ((and (near_uncharted_obstacle) (local_replan)) t)
    (t 1)))

  (defun noncritical_systems_OK ()
    (cond
      ((= *systems_abort* 1) t)
      ((CO-ask 'NonCritical_Systems_OK) t)
      (t nil) ))
    (defun global_replan()
      (cond
        ((= *systems_abort* 1) t)
        (t (CO-command 'Loiter_and_Start_Global_Replanner)) ))

    (defun near_uncharted_obstacle ()
      (cond
        ((= *systems_abort* 1) t)
        ((and (unknown_obstacle_p) (log_new_obstacle)) t)
        (t nil) ))

      (defun unknown_obstacle_p ()
        (cond
          ((= *systems_abort* 1) t)
          ((not (CO-ask 'Area_clear_of_uncharted_obstacles))
           (CO-command 'Log_new_obstacle))
          (t nil) ))

      (defun log_new_obstacle ()
        (cond
          ((= *systems_abort* 1) t)
          ((CO-ask 'New_obstacle_logged) t)
          ((CO-ask 'Log_system_failure)
           (and (setf *current_phase* 'mission_abort)
                (setf *systems_abort* 1) ))
          (t (log_new_obstacle)) ))

    (defun local_replan()
      (cond
        ((= *systems_abort* 1) t)
        (t (CO-command 'Loiter_and_Start_Local_Replanner)) ))


  (defun send_setpoints_and_modes ()
    (cond
      ((= *systems_abort* 1) t)
      ((CO-ask 'Setpoints_and_modes_sent) t)
      ((not (CO-ask 'Setpoints_and_modes_system_OK))
       (and (setf *current_phase* 'mission_abort)
            (setf *systems_abort* 1) ))
      (t (send_setpoints_and_modes)) ))


;----------- sonar_search doctrine block --------------------

(defun sonar_search ()
  (cond
    ((= *search_abort* 1) t)
    ((CO-ask 'Search_pattern_completed) t)
    ((CO-ask 'Sonar_failure)
     (and (setf *current_phase* 'mission_abort)
          (setf *search_abort* 1) ))
    (t (sonar_search)) ))
```

```
;---------------- do_task doctrine block --------------------

(defun do_task ()
  (and (homing)(drop_package)(get_gps_fix)) )


  (defun homing()
    (cond
      ((= *task_abort* 1) t)
              ((CO-ask  'Standoff_distance_reached)  (CO-command
'Drop_package))
      ((CO-ask 'Abort_homing)
       (and (setf *current_phase* 'mission_abort)
            (setf *task_abort* 1) ))
      (t (homing)) ))

  (defun drop_package()
    (cond
      ((= *task_abort* 1) t)
      ((CO-ask 'Is_package_dropped) (CO-command 'Get_GPS_fix))
      ((CO-ask 'Is_package_drop_aborted)
       (and (setf *current_phase* 'mission_abort)
            (setf *task_abort* 1) ))
      (t (drop_package)) ))

end co.cl
```

D. SUBORDINATE OFFICER CLASSES

Lisp File "ood.cl" contains the Engineer, Navigator, and Weapons Officer classes and Methods.

```
;-------------- ENGINEER CLASS AND METHODS  ------------------
(defclass engineerclass()() )

(defmethod order((engineer engineerclass)string)
    (write-string "          ENG --> EOOW : ")
    (write string)
    (write-line "!"))

(defmethod query((engineer engineerclass)string)
    (write-string "          ENG --> EOOW :  ")
    (write string)
    (write-string "?")
    (write-string " ")
    (member (read) *affirmative*))


;--------------- NAVIGATOR CLASS AND METHODS -----------------
(defclass navigatorclass()() )

(defmethod order((navigator navigatorclass)string)
    (write-string "          NAV --> QMOW : ")
    (write string)
    (write-line "!"))

(defmethod query((navigator navigatorclass)string)
    (write-string "          NAV --> QMOW : ")
    (write string)
    (write-string "?")
    (write-string " ")
    (member (read) *affirmative*))


;---------------- WEAPONS OFFICER CLASS AND METHODS ----------
(defclass weapons-officerclass()() )

(defmethod order((weapons-officer weapons-officerclass)string)
    (write-string "          WEPS --> SONAR: ")
    (write string)
    (write-line "!"))

(defmethod query((weapons-officer weapons-officerclass)string)
    (write-string "          WEPS --> SONAR: ")
    (write string)
    (write-string "?")
    (write-string " ")
    (member (read) *affirmative*))
```

```
;--------------- OOD CLASS AND METHODS -----------------------
(defclass oodclass ()

   ( (engineer
      :initform (make-instance 'engineerclass)
      :accessor engineer)
     (navigator
      :initform (make-instance 'navigatorclass)
      :accessor navigator)
     (weapons-officer
      :initform (make-instance 'weapons-officerclass)
      :accessor weapons-officer) ))


(defmethod command((ood oodclass) string person)
    (write-string "     OOD --> ")
    (write person)
    (write-string " :")
    (write string)
    (write-line "!")
    (order person string) )

(defmethod ask ((ood oodclass) string person)
    (write-string "     OOD --> ")
    (write person)
    (write-string " :")
    (write string)
    (write-line "?")
    (query person string) )

(setf *affirmative* '(1 t y))


;------------------ OOD ORDERS BLOCK -----------------------
(defmethod order((ood oodclass) order)

   (case order
      (Initialize_vehicle
          (and (command ood order (engineer ood))
               (command ood order (navigator ood) )
               (command ood order (weapons-officer ood)) ))

      (Transit_to_task_location
               (command ood order (navigator ood) ))

      (Search_area_for_target
               (command ood order (weapons-officer ood)))

      (Commence_task_on_target
               (command ood order (weapons-officer ood)))

      (Transit_to_recovery_point
                (command ood order (navigator ood) ))

      (Abort_mission
          (cond
                ((ask ood 'Is_recovery_point_obtainable
                 (navigator ood))
                 (setf *current_phase* 8))
                (t (command ood order (engineer ood)))))

      (Mission_complete
                (command ood order (engineer ood)))

      (Get_next-waypoint
                 (command ood order (navigator ood)))

      (Get_GPS_fix
                 (command ood order (navigator ood)))

      (Loiter_and_Start_Global_Replanner
                 (command ood order (navigator ood)))

      (Log_new_obstacle
                 (command ood order (navigator ood)))

      (Loiter_and_Start_Local_Replanner
                 (command ood order (navigator ood)))

      (Drop_package
                 (command ood order (weapons-officer ood)))

      (t nil) ))


;---------------- OOD QUERIES ----------------------------
(defmethod query ((ood oodclass) question)

   (case question

     (Critical_Systems_OK
       (cond ((ask ood question (engineer ood))
                   (write-line "OOD -->CO: yes" ) t)
             (t (write-line "OOD -->CO: no")) ))

     (Initialization_complete
       (cond ((and (ask ood question (weapons-officer ood))
                   (ask ood question (navigator ood))
                   (ask ood question (engineer ood))
                   (write-line "OOD -->CO: yes" )) t)
             (t (write-line "OOD -->CO: no")) ))

     (Initialization_aborted
       (cond ((and (not (ask ood question (weapons-officer ood)))
                   (not (ask ood question (navigator ood)))
                   (not (ask ood question (engineer ood)))
                   (write-line "OOD -->CO: no" )))
             (t (write-line "OOD -->CO: yes") t)))
```

```
(Task_location_reached
  (cond ((ask ood question (navigator ood))
         (write-line "OOD -->CO: yes" ) t)
        (t (write-line "OOD -->CO: no")) ))

(Target_found
  (cond ((ask ood question (weapons-officer ood))
         (write-line "OOD -->CO: yes" ) t)
        (t (write-line "OOD -->CO: no")) ))

(At_recovery_point
  (cond ((ask ood question (navigator ood))
         (write-line "OOD -->CO: yes" ) t)
        (t (write-line "OOD -->CO: no")) ))

(Waypoint_reached
  (cond ((ask ood question (navigator ood))
         (write-line "OOD -->CO: yes" ) t)
        (t (write-line "OOD -->CO: no")) ))

(Waypoint_process_OK
  (cond ((ask ood question (navigator ood))
         (write-line "OOD -->CO: yes" ) t)
        (t (write-line "OOD -->CO: no")) ))

(Got_next_waypoint
  (cond ((ask ood question (navigator ood))
         (write-line "OOD -->CO: yes" ) t)
        (t (write-line "OOD -->CO: no")) ))

(GPS_fix_needed
  (cond ((ask ood question (navigator ood))
         (write-line "OOD -->CO: yes" ) t)
        (t (write-line "OOD -->CO: no")) ))

(GPS_fix_obtained
  (cond ((ask ood question (navigator ood))
         (write-line "OOD -->CO: yes" ) t)
        (t (write-line "OOD -->CO: no")) ))

(Abort_GPS_fix
  (cond ((ask ood question (navigator ood))
         (write-line "OOD -->CO: yes" ) t)
        (t (write-line "OOD -->CO: no")) ))

(NonCritical_Systems_OK
  (cond ((and (ask ood question (weapons-officer ood))
              (ask ood question (engineer ood)) )
         (write-line "OOD -->CO: yes" ) t)
        (t (write-line "OOD -->CO: no")) ))

(Area_clear_of_uncharted_obstacles
  (cond ((ask ood question (navigator ood))
         (write-line "OOD -->CO: yes" ) t)
        (t (write-line "OOD -->CO: no")) ))
```

```
(New_obstacle_logged
  (cond ((ask ood question (navigator ood))
         (write-line "OOD -->CO: yes" ) t)
        (t (write-line "OOD -->CO: no")) ))

(Log_system_failure
  (cond ((ask ood question (navigator ood))
         (write-line "OOD -->CO: yes" ) t)
        (t (write-line "OOD -->CO: no")) ))

(Setpoints_and_modes_sent
  (cond ((ask ood question (navigator ood))
         (write-line "OOD -->CO: yes" ) t)
        (t (write-line "OOD -->CO: no")) ))

(Setpoints_and_modes_system_OK
  (cond ((ask ood question (navigator ood))
         (write-line "OOD -->CO: yes" ) t)
        (t (write-line "OOD -->CO: no")) ))

(Search_pattern_completed
  (cond ((ask ood question (weapons-officer ood))
         (write-line "OOD -->CO: yes" ) t)
        (t (write-line "OOD -->CO: no")) ))

(Sonar_failure
  (cond ((ask ood question (weapons-officer ood))
         (write-line "OOD -->CO: yes" ) t)
        (t (write-line "OOD -->CO: no")) ))

(Standoff_distance_reached
  (cond ((ask ood question (weapons-officer ood))
         (write-line "OOD -->CO: yes" ) t)
        (t (write-line "OOD -->CO: no")) ))

(Abort_homing
  (cond ((ask ood question (weapons-officer ood))
         (write-line "OOD -->CO: yes" ) t)
        (t (write-line "OOD -->CO: no")) ))

(Is_package_dropped
  (cond ((ask ood question (weapons-officer ood))
         (write-line "OOD -->CO: yes" ) t)
        (t (write-line "OOD -->CO: no")) ))

(Is_package_drop_aborted
  (cond ((ask ood question (weapons-officer ood))
         (write-line "OOD -->CO: yes" ) t)
        (t (write-line "OOD -->CO: no")) ))

(t nil) ))
```

APPENDIX D. STRATEGIC AND TACTICAL LEVEL ADA CODE

This appendix contains the Ada code for the Strategic and Tactical Levels of the RBM.

A. MISSION FILE

Phase	Action	Phase to go to if Succeed	Phase to go to if Fail	Step Type
1	Initialize vehicle	2	98	1
2	Transit to task location	3	2	2
3	Search area for target	4	3	3
4	Report targets found	5	6	4
5	Commence task on target	6	5	5
6	Transit to task location	7	6	2
7	Search area for target	8	7	3
8	Report targets found	9	10	4
9	Commence task on target	10	9	5
10	Transit to recovery point	99	10	11
98	Abort Mission(98)	98	98	12
99	Mission Complete(99)	99	99	13

B. STRATEGIC LEVEL - FILE "co_auv.a"

```
-------------------- co_auv.a -----------------------------
-- Title          : AUV Ada Commanding Officer program body
-- Filename       : co_auv.a
-- Author         : Michael J. Holden
-- Date           : 15 May 1995        Revised: June-July 1995
-- Course         : Thesis Work
-- Compiler       : SunAda SunOS Release 4.1.3
-------------------- co_auv.a -----------------------------

  with TEXT_IO;      use TEXT_IO;
  with MY_INT_IO;    use MY_INT_IO;
  with DOCTRINE_AUV; use DOCTRINE_AUV;
  with WARDROOM_AUV; use WARDROOM_AUV;

procedure CO_AUV is
  NameLength     : Natural;  -- Holds length of FileName string
```

```
 FileName      : String (1..12);     -- Holds user-supplied file name
 Phase        : Integer :=1;

 type PTR_TO_OFFICER is access WARDROOM_AUV.OFFICER;
 OOD: PTR_TO_OFFICER;

begin -- CO_AUV

 OOD  := new WARDROOM_AUV.OFFICER;

 Put("Enter mission file name =>  ");
 Get_Line(FileName, NameLength);

 New_Line;
 Put(Phase, Width=>3); Put(".  ");
 Put_Line("CO --> OOD: Get Mission Orders!");
 Put(Phase, Width=>3); Put(".  ");
 Put("  OOD --> CO: ");
 OOD.ORDER("Get Mission Orders");

 INITIALIZE_MISSION(Filename(1..NameLength));
 while not (DONE(Phase)) loop
  EXECUTE_PHASE(Phase);
 end loop;

end CO_AUV;
```

C. TACTICAL LEVEL - FILE “wardroom_auv.a”

```
-------------------- wardroom_auv.a -----------------------------
-- Title          : AUV Officer program body
-- Filename       : wardroom_auv.a
-- Author         : Michael J. Holden
-- Date           : 15 May 1995          Revised: June-July 1995
-- Course         : Thesis Work
-- Compiler       : SunAda SunOS Release 4.1.3
-------------------- wardroom_auv.a -----------------------------

 with TEXT_IO;   use TEXT_IO;
 with MY_INT_IO; use MY_INT_IO;

 package WARDROOM_AUV is
  task type OFFICER is
   entry order(Report1 : in String);
   entry report(Report2 : out Boolean);
  end OFFICER;
 end WARDROOM_AUV;

 package body WARDROOM_AUV is
  task body OFFICER is
   ANSWER: CHARACTER;
   begin
    loop
     select
      accept order(Report1 : in String) do
       Put_Line(Report1(1..Report1'Length) & “, Aye sir!”);
      end order;
     or
      accept report(Report2 : out Boolean) do
       Put(“Answer => “);
       Get(ANSWER);
       if (ANSWER = ‘y’) then Report2 := true;
       else Report2 := false;
       end if;
      end report;
     or
      terminate;
     end select;
    end loop;
  end OFFICER;
 end WARDROOM_AUV;
```

D. TACTICAL LEVEL - FILE "doctrine_auv.a"

```
-------------------- doctrine_auv.a -----------------------------
-- Title          : AUV Ada Doctrine package
-- Filename       : doctrine_auv.a
-- Author         : Michael J. Holden
-- Date           : 15 May 1995      Revised: June-July 1995
-- Course         : Thesis Work
-- Compiler       : SunAda SunOS Release 4.1.3
-- Description    : Breaks out separate officer objects for Engineer, Navigator,
--                  Weapons Officer
-------------------- doctrine_auv.a -----------------------------

 with Text_IO; use Text_IO;
 with MY_INT_IO; use MY_INT_IO;
 with WARDROOM_AUV; use WARDROOM_AUV;
 with ENGINEER; use ENGINEER;
 with NAVIGATOR; use NAVIGATOR;
 with WEAPONSOFFICER; use WEAPONSOFFICER;

package DOCTRINE_AUV is

 procedure INITIALIZE_MISSION(MyName: in String);
 procedure EXECUTE_PHASE(Phase: in out Integer);
 procedure ISSUE_ORDERS(Phase: in out Integer);
 procedure PHASE_COMPLETED(Phase: in out Integer);
 procedure NEXT_PHASE(Phase: in out Integer);
 function DONE(Phase: in Integer) return Boolean;

 procedure SONAR_SEARCH(Phase: in out Integer);

 procedure PERFORM_TASK(Phase: in out Integer);

 procedure WAYPOINT_CONTROL(Phase: in out Integer);
 procedure GET_WAYPOINT_STATUS(Phase: in out Integer);
 procedure GPS_CHECK(Phase: in out Integer);
 procedure PLAN(Phase: in out Integer);
 procedure SEND_SETPOINTS_AND_MODES(Phase: in out Integer);

end DOCTRINE_AUV;

package body DOCTRINE_AUV is

 InData       : Text_IO.File_Type;   -- Incoming data file
```

```
NameLength    : Natural;           -- Holds length of FileName string
PhaseComplete : Boolean := false;
Report2       : Boolean;

type PTR_TO_OFFICER is access WARDROOM_AUV.OFFICER;
type PTR_TO_ENGINEER is access ENGINEER.OFFICER;
type PTR_TO_NAVIGATOR is access NAVIGATOR.OFFICER;
type PTR_TO_WEAPONSOFFICER is access WEAPONSOFFICER.OFFICER;

WEPS: PTR_TO_WEAPONSOFFICER;
NAV: PTR_TO_NAVIGATOR;
ENG: PTR_TO_ENGINEER;

type MissionPhase is record
 PHASE          : Integer;
 NAME                : String(1..31);
 SUCCEED          : Integer;
 FAIL                 : Integer;
 STEPTYPE  : Integer;
end record;

type Array_of_Records is array (NATURAL range <>) of MissionPhase;

type TABLE(TABLE_SIZE : natural) is record
 OBJECTS       : Array_of_Records(1..TABLE_SIZE);
end record;

MissionOrders:  TABLE(100);  -- maximum number of mission phases

------------------- procedures -----------------------------

procedure INITIALIZE_MISSION(MyName: in String) is

i : integer := 1;

begin -- INITIALIZE_MISSION
 -- goal is to have a changeable mission file so that
 -- the mission phases can be dynamically changed

 ENG  := new ENGINEER.OFFICER;
 NAV  := new NAVIGATOR.OFFICER;
 WEPS := new WEAPONSOFFICER.OFFICER;
 Text_IO.New_Line;
 Put_Line("Phase    Action                  Next Phase to go to  ");
 Put_Line("----------------------------------------------------------");
 Open(File=>InData, Mode=>Text_IO.In_File, Name=>MyName);
 while not Text_IO.End_of_File(File => InData) loop
   Get(File => InData, Item =>MissionOrders.Objects(i).PHASE);
   Get(File => InData, Item =>MissionOrders.Objects(i).NAME);
   Get(File => InData, Item =>MissionOrders.Objects(i).SUCCEED);
   Get(File => InData, Item =>MissionOrders.Objects(i).FAIL);
   Get(File => InData, Item =>MissionOrders.Objects(i).STEPTYPE);
   Put(i, Width=>3);
   Put(MissionOrders.Objects(i).NAME);
   Put("  If Succeed/Fail = ");
   Put(MissionOrders.Objects(i).SUCCEED, Width=>3);
   Put("/");
   Put(MissionOrders.Objects(i).FAIL, Width=>3); New_Line;
   i:= i+1;
 end loop;
 Put_Line("----------------------------------------------------------");
 -- Close(File => InData);
end INITIALIZE_MISSION;

procedure EXECUTE_PHASE(Phase: in out INTEGER) is
  -- checks for critical systems problems (aborts if they exist)
  -- then issues orders and checks for completion of the current phase
  -- then moves on to the next phase
  ReportE : Boolean := false;
begin -- EXECUTE_PHASE
  Put(Phase, Width=>3); Put(".  ");
  Put_Line("CO --> OOD: Are Critical Systems OK?");
  Put(Phase, Width=>3); Put(".  ");
  Put("  OOD  --> ENG: ");
  Put("Are Critical Systems OK, Eng? ");
  ENG.REPORT(ReportE);
  if not ReportE then
    Put(Phase, Width=>3); Put(".  ");
    Put_Line("  OOD --> CO: Critical Systems Failure, Sir.");
    Phase := 98;
  else
    Put(Phase, Width=>3); Put(".  ");
    Put_Line("  OOD --> CO: Critical Systems OK, Sir.");
    ISSUE_ORDERS(Phase);
    PHASE_COMPLETED(Phase);
    NEXT_PHASE(Phase);
   end if;
  end EXECUTE_PHASE;

  procedure ISSUE_ORDERS(Phase: in out INTEGER) is
  begin -- ISSUE_ORDERS
    -- READ in from mission file what the order for this phase is
--    Put_Line("Issue Orders. ");
    Put(Phase, Width=>3); Put(".  ");
    Put("CO --> OOD: ");
    case missionOrders.Objects(Phase).STEPTYPE  is
      when 1   => Put_Line("Initialize! ");
                  Put(Phase, Width=>3); Put(".  ");
                  Put("  OOD --> NAV, ENG, WEPS: ");
                  Put_Line("Initialize! ");
                  Put(Phase, Width=>3); Put(".  ");
                  Put("    NAV --> OOD: ");
                  NAV.ORDER("Initialize");
                  Put(Phase, Width=>3); Put(".  ");
                  Put("    ENG --> OOD: ");
                  ENG.ORDER("Initialize");
                  Put(Phase, Width=>3); Put(".  ");
                  Put("    WEPS --> OOD: ");
                  WEPS.ORDER("Initialize");
      when 2   => Put_Line("Transit! ");
                  Put(Phase, Width=>3); Put(".  ");
                  Put("  OOD --> NAV: ");
                  Put_Line("Transit! ");
                  Put(Phase, Width=>3); Put(".  ");
                  Put("    NAV --> OOD: ");
                  NAV.ORDER("Transit");
                  WAYPOINT_CONTROL(Phase);
      when 3   => Put_Line("Search! ");
                  Put(Phase, Width=>3); Put(".  ");
                  Put("  OOD --> WEPS: ");
                  Put_Line("Search! ");
                  Put(Phase, Width=>3); Put(".  ");
                  Put("    WEPS --> OOD: ");
                  SONAR_SEARCH(Phase);
      when 4   => Put_Line("Report targets found! ");
                  Put(Phase, Width=>3); Put(".  ");
                  Put("  OOD --> WEPS: ");
                  Put_Line("Report targets found! ");
```

```
            Put(Phase, Width=>3); Put(". ");
            Put("   WEPS --> OOD: ");
            WEPS.ORDER("Report targets found");
    when 5   => Put_Line("Commence task on Target! ");
            Put(Phase, Width=>3); Put(". ");
            Put("  OOD --> WEPS: ");
            Put_Line("Commence task on Target! ");
            Put(Phase, Width=>3); Put(". ");
            Put("   WEPS --> OOD: ");
            WEPS.ORDER("Commence task on Target");
            PERFORM_TASK(Phase);
    when 11  => Put_Line("Transit to Recovery Point! ");
            Put(Phase, Width=>3); Put(". ");
            Put("  OOD --> NAV: ");
            Put_Line("Transit to Recovery Point! ");
            Put(Phase, Width=>3); Put(". ");
            Put("   NAV --> OOD: ");
            NAV.ORDER("Transit to Recovery Point");
    when 12  => Put_Line("Mission Abort!");
    when 13  => Put_Line("Mission Complete! ");
    when others =>Put_Line("Unknown order");
  end case;
end ISSUE_ORDERS;

procedure PHASE_COMPLETED(Phase: in out INTEGER) is
  ReportN : Boolean := false;
  ReportE : Boolean := false;
  ReportW : Boolean := false;
begin -- PHASE_COMPLETED
--   Put_Line("Check for phase completion. ");
  Put(Phase, Width=>3); Put(". ");
  Put("CO --> OOD: ");
  case missionOrders.Objects(Phase).STEPTYPE  is
    when 1   => Put_Line("Initialization Complete? ");
            Put(Phase, Width=>3); Put(". ");
            Put("  OOD  --> NAV: ");
            Put("Initialization Complete, Nav? ");
            NAV.REPORT(ReportN);
            Put(Phase, Width=>3); Put(". ");
            Put("  OOD  --> ENG: ");
            Put("Initialization Complete, Eng? ");
            ENG.REPORT(ReportE);
            Put(Phase, Width=>3); Put(". ");
```

```
          Put("  OOD  --> WEPS: ");
          Put("Initialization Complete, Weps? ");
          WEPS.REPORT(ReportW);
          if ReportN and ReportE and ReportW then
            PhaseComplete := true;
            Put(Phase, Width=>3); Put(".  ");
            Put_Line("  OOD  --> CO: Initialization Complete, Sir.");
          else
            PhaseComplete := false;
            Put(Phase, Width=>3); Put(".  ");
            Put_Line("  OOD  --> CO: Initialization failure, Sir. ");
          end if;

  when 2   => Put_Line("Transit complete? ");
          Put(Phase, Width=>3); Put(".  ");
          Put("  OOD  --> NAV: ");
          Put("Transit complete? ");
          NAV.REPORT(ReportN);
          if ReportN then
            PhaseComplete := true;
            Put(Phase, Width=>3); Put(".  ");
            Put_Line("  OOD  --> CO: Transit Complete, Sir.");
          else
            Put(Phase, Width=>3); Put(".  ");
            Put_Line("  OOD  --> CO: Transit incomplete, Sir. ");
            PhaseComplete := false;
          end if;

  when 3   => Put_Line("Search Pattern Complete? ");
          Put(Phase, Width=>3); Put(".  ");
          Put("  OOD --> WEPS: ");
          Put("Search Pattern Complete? ");
          WEPS.REPORT(ReportW);
          if ReportW then
            PhaseComplete := true;
            Put(Phase, Width=>3); Put(".  ");
            Put_Line("  OOD  --> CO: Search Pattern Complete, Sir.");
          else
            PhaseComplete := false;
            Put(Phase, Width=>3); Put(".  ");
            Put_Line("  OOD  --> CO: Search Pattern incomplete, Sir.");
          end if;
```

```
when 4  => Put_Line("Any targets found? ");
        Put(Phase, Width=>3); Put(".  ");
        Put("  OOD --> WEPS: ");
        Put("Any targets found? ");
        WEPS.REPORT(ReportW);
        if ReportW then
          PhaseComplete := true;
          Put(Phase, Width=>3); Put(".  ");
          Put_Line("  OOD  --> CO: Targets in search area, Sir.");
        else
          PhaseComplete := false;
          Put(Phase, Width=>3); Put(".  ");
          Put_Line("  OOD  --> CO: No targets in search area, Sir.");
        end if;

when 5  => Put_Line("Task on Target Complete? ");
        Put(Phase, Width=>3); Put(".  ");
        Put("  OOD --> WEPS: ");
        Put("Task on Target Complete? ");
        WEPS.REPORT(ReportW);
        if ReportW then
          PhaseComplete := true;
          Put(Phase, Width=>3); Put(".  ");
          Put_Line("  OOD  --> CO: Task Complete, Sir.");
        else
          PhaseComplete := false;
          Put(Phase, Width=>3); Put(".  ");
          Put_Line("  OOD  --> CO: Task incomplete, Sir.");
        end if;

when 11  => Put_Line("At Recovery Point? ");
        Put(Phase, Width=>3); Put(".  ");
        Put("  OOD  --> NAV: ");
        Put("At Recovery Point? ");
        NAV.REPORT(ReportN);
        if ReportN then
          PhaseComplete := true;
          Put(Phase, Width=>3); Put(".  ");
          Put_Line("  OOD  --> CO: Transit Complete, Sir.");
        else
          PhaseComplete := false;
          Put_Line("  OOD  --> CO: Transit incomplete, Sir.");
        end if;
```

```
    when 12  => Put_Line("Mission Abort! ");
    when 13  => Put_Line("Mission Complete! ");
    when others => Put_Line("Unknown Order. ");
   end case;
  end PHASE_COMPLETED;

  procedure NEXT_PHASE(Phase: in out INTEGER) is
  begin -- NEXT_PHASE
--   Put_Line("Go to next phase. ");
    New_Line;
     if (PhaseComplete) then
       Phase := MissionOrders.Objects(Phase).SUCCEED;
     else
       Phase := MissionOrders.Objects(Phase).FAIL;
     end if;
  end NEXT_PHASE;

  function DONE(Phase: in INTEGER) return BOOLEAN is
  begin -- DONE
   New_Line;
--   Put_Line("Check for mission completion or mission abort.");
   case Phase is
    when 99          =>
     Put("     ");
     Put_Line("CO --> OOD: Mission Complete!");
     return true;
    when 98  =>
     Put("     ");
     Put_Line("CO --> OOD: Mission Aborted!");
     return true;
    when others => return false;
   end case;
  end DONE;

---------------------------- sonar search block ---------------------

  procedure SONAR_SEARCH(Phase: in out Integer) is
   AbortSearch: Boolean := false;
   SearchComplete: Boolean := false;
  begin -- SONAR_SEARCH
   WEPS.ORDER("Search");
   Put(Phase, Width=>3); Put(". ");
  Put("  OOD --> WEPS: ");
  Put("Search Complete, Weps? ");
  WEPS.REPORT(SearchComplete);
  if not SearchComplete then
   Put(Phase, Width=>3); Put(".  ");
   Put("  OOD --> WEPS: ");
   Put("Abort Search, Weps? ");
   WEPS.REPORT(AbortSearch);
  end if;
  if AbortSearch then
  Put(Phase, Width=>3); Put(".  ");
  Put_Line("  OOD --> CO: Search System Failure, Sir.");
  missionOrders.Objects(Phase).SUCCEED:= 98;
  missionOrders.Objects(Phase).FAIL:= 98;
  end if;
 end SONAR_SEARCH;

---------------------------- task block --------------------

 procedure PERFORM_TASK(Phase: in out Integer) is
  StandoffDistanceReached : Boolean := false;
  AbortHoming : Boolean := false;
  AbortDrop : Boolean := false;
  DropComplete : Boolean := false;
 begin -- PERFORM_TASK
  Put(Phase, Width=>3); Put(".  ");
  Put("  OOD --> WEPS: ");
  Put("Standoff Distance Reached, Weps? ");
  WEPS.REPORT(StandoffDistanceReached);
  if StandoffDistanceReached then
   Put(Phase, Width=>3); Put(".  ");
   Put("  OOD --> WEPS: ");
   Put_Line("Drop Package! ");
   Put(Phase, Width=>3); Put(".  ");
   Put("    WEPS --> OOD: ");
   WEPS.ORDER("Drop Package");
   Put(Phase, Width=>3); Put(".  ");
   Put("  OOD --> WEPS: ");
   Put("Package Drop Complete, Weps? ");
   WEPS.REPORT(DropComplete);
   if not DropComplete then
    Put(Phase, Width=>3); Put(".  ");
    Put("  OOD --> WEPS: ");
```

```
      Put("Abort Drop, Weps? ");
      WEPS.REPORT(AbortDrop);
      if AbortDrop then
        Put(Phase, Width=>3); Put(".  ");
        Put_Line("  OOD --> CO: Package Drop System Failure, Sir.");
        missionOrders.Objects(Phase).SUCCEED:= 98;
        missionOrders.Objects(Phase).FAIL:= 98;
      end if;
    end if;
  else
    Put(Phase, Width=>3); Put(".  ");
    Put("  OOD --> WEPS: ");
    Put("Abort Homing, Weps? ");
    WEPS.REPORT(AbortHoming);
  end if;
  if AbortHoming then
    Put(Phase, Width=>3); Put(".  ");
    Put_Line("  OOD --> CO: Homing System Failure, Sir.");
    missionOrders.Objects(Phase).SUCCEED:= 98;
    missionOrders.Objects(Phase).FAIL:= 98;
  end if;
  GPS_CHECK(Phase);
end PERFORM_TASK;

---------------------------- waypoint control block ---------------------

procedure WAYPOINT_CONTROL(Phase: in out Integer) is
begin -- WAYPOINT_CONTROL
  GET_WAYPOINT_STATUS(Phase);
  PLAN(Phase);
  SEND_SETPOINTS_AND_MODES(Phase);
end WAYPOINT_CONTROL;

procedure GET_WAYPOINT_STATUS(Phase: in out Integer) is
  WaypointReached   : Boolean := false;
  WaypointComplete : Boolean := false;
  AbortWaypoint     : Boolean := false;
begin -- GET_WAYPOINT_STATUS
  Put(Phase, Width=>3); Put(".  ");
  Put("  OOD --> NAV: ");
  Put("Waypoint Reached, Nav? ");
  NAV.REPORT(WaypointReached);
  if WaypointReached then
```

```
    Put(Phase, Width=>3); Put(". ");
    Put("  OOD --> NAV: ");
    Put_Line("Get Next Waypoint! ");
    Put(Phase, Width=>3); Put(". ");
    Put("    NAV --> OOD: ");
    NAV.ORDER("Get Next Waypoint");
    Put(Phase, Width=>3); Put(". ");
    Put("  OOD --> NAV: ");
    Put("Got Next Waypoint, Nav? ");
    NAV.REPORT(WaypointComplete);
    if not WaypointComplete then
      Put(Phase, Width=>3); Put(". ");
      Put("  OOD --> NAV: ");
      Put("Abort Waypoint Process, Nav? ");
      NAV.REPORT(AbortWaypoint);
      if AbortWaypoint then
        Put(Phase, Width=>3); Put(". ");
        Put_Line("  OOD --> CO: Waypoint Process System Failure, Sir.");
        missionOrders.Objects(Phase).SUCCEED:= 98;
        missionOrders.Objects(Phase).FAIL:= 98;
      end if;
    end if;
    GPS_CHECK(Phase);
  end if;
end GET_WAYPOINT_STATUS;

procedure GPS_CHECK(Phase: in out Integer) is
  GPSFixNeeded   : Boolean := false;
  GPSFixObtained : Boolean := false;
  AbortGPSFix    : Boolean := false;
begin -- GPS_CHECK
  Put(Phase, Width=>3); Put(". ");
  Put("  OOD --> NAV: ");
  Put("GPS Fix Needed, Nav? ");
  NAV.REPORT(GPSFixNeeded);
  if GPSFixNeeded then
    Put(Phase, Width=>3); Put(". ");
    Put("  OOD --> NAV: ");
    Put_Line("Get GPS Fix! ");
    Put(Phase, Width=>3); Put(". ");
    Put("    NAV --> OOD: ");
    NAV.ORDER("Get GPS Fix");
    Put(Phase, Width=>3); Put(". ");
```

```
   Put("  OOD --> NAV: ");
   Put("GPS Fix Obtained, Nav? ");
   NAV.REPORT(GPSFixObtained );
   if not GPSFixObtained then
     Put(Phase, Width=>3); Put(".  ");
     Put("  OOD --> NAV: ");
     Put("Abort GPS Fix, Nav? ");
     NAV.REPORT(AbortGPSFix);
     if AbortGPSFix then
       Put(Phase, Width=>3); Put(".  ");
       Put_Line("  OOD --> CO: GPS Failure, Sir.");
       missionOrders.Objects(Phase).SUCCEED:= 98;
       missionOrders.Objects(Phase).FAIL:= 98;
     end if;
   end if;
  end if;
end GPS_CHECK;

procedure PLAN(Phase: in out Integer) is
  NonCriticalSystemsOK : Boolean := false;
  NearUnchartedObstacle: Boolean := false;
  ReplanningComplete   : Boolean := false;
  AbortPlan            : Boolean := false;
begin -- PLAN
  Put(Phase, Width=>3); Put(".  ");
  Put("  OOD --> ENG: ");
  Put("Are NonCritical Systems OK, Eng? ");
  ENG.REPORT(NonCriticalSystemsOK);
  Put(Phase, Width=>3); Put(".  ");
  Put("  OOD --> NAV: ");
  Put("Any Uncharted Obstacles, Nav? ");
  NAV.REPORT(NearUnchartedObstacle);
  if not NonCriticalSystemsOK then
    Put(Phase, Width=>3); Put(".  ");
    Put("  OOD --> NAV: ");
    Put_Line("Commence Global Replanning! ");
    Put(Phase, Width=>3); Put(".  ");
    Put("    NAV --> OOD: ");
    NAV.ORDER("Commence Global Replanning, Nav");
  elsif NearUnchartedObstacle then
    Put(Phase, Width=>3); Put(".  ");
    Put("  OOD --> NAV: ");
    Put_Line("Commence Local Replanning! ");
  Put(Phase, Width=>3); Put(". ");
  Put("    NAV --> OOD: ");
  NAV.ORDER("Commence Local Replanning, Nav");
 end if;
 if ((not NonCriticalSystemsOK) or NearUnchartedObstacle) then
  Put(Phase, Width=>3); Put(". ");
  Put("  OOD --> NAV: ");
  Put("Replanning Complete, Nav? ");
  NAV.REPORT(ReplanningComplete);
  if not ReplanningComplete then
   Put(Phase, Width=>3); Put(". ");
   Put("  OOD --> NAV: ");
   Put("Abort Replanning, Nav? ");
   NAV.REPORT(AbortPlan);
   if AbortPlan then
    Put(Phase, Width=>3); Put(". ");
    Put_Line("  OOD --> CO: Replanning Systems Failure, Sir.");
    missionOrders.Objects(Phase).SUCCEED:= 98;
    missionOrders.Objects(Phase).FAIL:= 98;
   end if;
  end if;
 end if;
end PLAN;

procedure SEND_SETPOINTS_AND_MODES(Phase: in out Integer) is
 SetpointsSent       : Boolean := false;
 AbortNavSystem      : Boolean := false;
begin -- SEND_SETPOINTS_AND_MODES
 Put(Phase, Width=>3); Put(". ");
 Put("  OOD --> NAV: ");
 Put_Line("Send Setpoints and Modes!");
 Put(Phase, Width=>3); Put(". ");
 Put("    NAV --> OOD: ");
 NAV.ORDER("Send Setpoints and Modes, Nav");
 Put(Phase, Width=>3); Put(". ");
 Put("  OOD --> NAV: ");
 Put("Setpoints sent, Nav? ");
 NAV.REPORT(SetpointsSent);
 if not SetpointsSent then
  Put(Phase, Width=>3); Put(". ");
  Put("  OOD --> NAV: ");
  Put("Navigation System Failure, Nav? ");
  NAV.REPORT(AbortNavSystem);
```

```
    if AbortNavSystem then
      Put(Phase, Width=>3); Put(". ");
      Put_Line("  OOD --> CO: Navigation Systems Failure, Sir.");
      missionOrders.Objects(Phase).SUCCEED:= 98;
      missionOrders.Objects(Phase).FAIL:= 98;
    end if;
  end if;
end SEND_SETPOINTS_AND_MODES;

-- procedure NONCRITICAL_SYSTEMS_OK(Phase: in out Integer) is
-- begin -- NONCRITICAL_SYSTEMS_OK
--   Put_Line("Check NonCritical Systems");
-- end NONCRITICAL_SYSTEMS_OK;

-- procedure GLOBAL_REPLAN(Phase: in out Integer) is
-- begin -- GLOBAL_REPLAN
--   Put_Line("Do Global Replan");
-- end GLOBAL_REPLAN;

-- procedure NEAR_UNCHARTED_OBSTACLE(Phase: in out Integer) is
-- begin -- NEAR_UNCHARTED_OBSTACLE
--   Put_Line("Near Uncharted Obstacle");
-- end NEAR_UNCHARTED_OBSTACLE;

-- procedure UNKNOWN_OBSTACLE(Phase: in out Integer) is
-- begin -- UNKNOWN_OBSTACLE
--   Put_Line("Unknown Obstacle");
-- end UNKNOWN_OBSTACLE;

-- procedure LOG_NEW_OBSTACLE(Phase: in out Integer) is
-- begin -- LOG_NEW_OBSTACLE
--   Put_Line("Log New Obstacle");
-- end LOG_NEW_OBSTACLE;

-- procedure LOCAL_REPLAN(Phase: in out Integer) is
-- begin -- LOCAL_REPLAN
--   Put_Line("Do Local Replan");
-- end LOCAL_REPLAN;

end DOCTRINE_AUV;
```

E. TACTICAL LEVEL - FILE “eng.a”

```
----------------------- eng.a -----------------------------
-- Title          : AUV ENGINEER program body
-- Filename       : eng.a
-- Author         : Michael J. Holden
-- Date           : 15 May 1995        Revised: June-July 1995
-- Course         : Thesis Work
-- Compiler       : SunAda SunOS Release 4.1.3
-- Description    : Breaks out separate Engineer officer object
----------------------- eng.a -----------------------------

 with TEXT_IO;   use TEXT_IO;
 with MY_INT_IO; use MY_INT_IO;
 with WARDROOM_AUV; use WARDROOM_AUV;


 package ENGINEER is
  task type OFFICER is
    entry order(Report1 : in String);
    entry report(Report2 : out Boolean);
  end OFFICER;
 end ENGINEER;

 package body ENGINEER is

  type PTR_TO_OFFICER is access WARDROOM_AUV.OFFICER;
  EOOW : PTR_TO_OFFICER;

  task body OFFICER is
   ANSWER: CHARACTER;
   begin
    EOOW := new  WARDROOM_AUV.OFFICER;
    loop
     select
      accept order(Report1 : in String) do
       Put_Line(Report1(1..Report1'Length) & ", Aye sir!");
       Put("     ");
       Put("     ENG --> EOOW: ");
       Put_Line(Report1(1..Report1'Length));
       Put("            ");
       EOOW.ORDER("Initialize, EOOW");
      end order;
     or
```

```
          accept report(Report2 : out Boolean) do
            Put("Answer => ");
            Get(ANSWER);
            if (ANSWER = 'y') then Report2 := true;
            else Report2 := false;
            end if;
          end report;
        or
          terminate;
        end select;
      end loop;
  end OFFICER;
end ENGINEER;
```

F. TACTICAL LEVEL - FILE "weps.a"

```
----------------------- weps.a --------------------------------
-- Title          : AUV Officer program body
-- Filename       : weps.a
-- Author         : Michael J. Holden
-- Date           : 15 May 1995       Revised: June-July 1995
-- Course         : Thesis Work
-- Compiler       : SunAda SunOS Release 4.1.3
-- Description    : Code for separate weapons officer object
----------------------- weps.a --------------------------------

 with TEXT_IO;   use TEXT_IO;
 with MY_INT_IO; use MY_INT_IO;
 with WARDROOM_AUV; use WARDROOM_AUV;


 package WEAPONSOFFICER is
  task type OFFICER is
   entry order(Report1 : in String);
   entry report(Report2 : out Boolean);
  end OFFICER;
 end WEAPONSOFFICER;

 package body WEAPONSOFFICER is

  type PTR_TO_OFFICER is access WARDROOM_AUV.OFFICER;
  TMOW : PTR_TO_OFFICER;

  task body OFFICER is
   ANSWER: CHARACTER;
   begin
    TMOW := new  WARDROOM_AUV.OFFICER;
    loop
     select
      accept order(Report1 : in String) do
       Put_Line(Report1(1..Report1'Length) & ", Aye sir!");
       Put("     ");
       Put("     WEPS --> TMOW: ");
       Put_Line(Report1(1..Report1'Length)& "!");
       Put("           ");
       TMOW.ORDER((Report1(1..Report1'Length)& ", TMOW"));
      end order;
     or
```

```
          accept report(Report2 : out Boolean) do
            Put("Answer => ");
            Get(ANSWER);
            if (ANSWER = 'y') then Report2 := true;
            else Report2 := false;
            end if;
          end report;
        or
          terminate;
        end select;
      end loop;
  end OFFICER;
end WEAPONSOFFICER;
```

G. TACTICAL LEVEL - FILE "nav.a"

```
------------------------ nav.a ----------------------------------
-- Title            : AUV Officer program body
-- Filename         : nav.a
-- Author           : Michael J. Holden
-- Date             : 15 May 1995          Revised: June-July 1995
-- Course           : Thesis Work
-- Compiler         : SunAda SunOS Release 4.1.3
-- Description      : Code for separate navigator officer object
------------------------ nav.a ---------------------------------

 with TEXT_IO;   use TEXT_IO;
 with MY_INT_IO; use MY_INT_IO;
 with WARDROOM_AUV; use WARDROOM_AUV;


 package NAVIGATOR is
  task type OFFICER is
   entry order(Report1 : in String);
   entry report(Report2 : out Boolean);
  end OFFICER;
 end NAVIGATOR;

 package body NAVIGATOR is

  type PTR_TO_OFFICER is access WARDROOM_AUV.OFFICER;
  QMOW : PTR_TO_OFFICER;

  task body OFFICER is
   ANSWER: CHARACTER;
   begin
    QMOW := new  WARDROOM_AUV.OFFICER;
    loop
     select
      accept order(Report1 : in String) do
       Put_Line(Report1(1..Report1'Length) & ", Aye sir!");
       Put("      ");
       Put("      NAV --> QMOW: ");
       Put_Line(Report1(1..Report1'Length)& "!");
       Put("            ");
       QMOW.ORDER((Report1(1..Report1'Length)& ", QMOW"));
      end order;
     or
```

```
          accept report(Report2 : out Boolean) do
            Put(“Answer => “);
            Get(ANSWER);
            if (ANSWER = ‘y’) then Report2 := true;
            else Report2 := false;
            end if;
          end report;
        or
          terminate;
        end select;
      end loop;
  end OFFICER;
end NAVIGATOR;
```

APPENDIX E. PROGRAM EXECUTION TRACES

This appendix contains traces of LISP and Ada program execution for testing the Strategic and Tactical Levels of the Rational Behavior Model (RBM) .

A. LISP TRACES

1. LISP Trace - Fully Successful Mission

```
user(1): (load "ood.cl") (load "co.cl") (execute_mission)

; Loading /users/work2/mjholden/thesis/mythesis/LISP/ood.cl.
t
user(2):
; Loading /users/work2/mjholden/thesis/mythesis/LISP/co.cl.
t
user(3):

1.CO -->OOD: Critical_Systems_OK?
   OOD --> #<engineerclass @ #xcec8f6> :Critical_Systems_OK?
      ENG --> EOOW :  Critical_Systems_OK? y
OOD -->CO: yes

1.CO -->OOD: Initialize_vehicle!
   OOD --> #<engineerclass @ #xcec8f6> :Initialize_vehicle!
      ENG --> EOOW : Initialize_vehicle!
   OOD --> #<navigatorclass @ #xcebfee> :Initialize_vehicle!
      NAV --> QMOW : Initialize_vehicle!
   OOD --> #<weapons-officerclass @ #xceb2de> :Initialize_vehicle!
      WEPS --> SONAR: Initialize_vehicle!

1.CO -->OOD: Initialization_complete?
   OOD --> #<weapons-officerclass @ #xceb2de> :Initialization_complete?
      WEPS --> SONAR: Initialization_complete? y
   OOD --> #<navigatorclass @ #xcebfee> :Initialization_complete?
      NAV --> QMOW : Initialization_complete? y
   OOD --> #<engineerclass @ #xcec8f6> :Initialization_complete?

OOD -->CO: yes
```

```
2.CO -->OOD: Critical_Systems_OK?
   OOD --> #<engineerclass @ #xcec8f6> :Critical_Systems_OK?
      ENG --> EOOW :  Critical_Systems_OK? y
OOD -->CO: yes

2.CO -->OOD: Transit_to_task_location!
   OOD --> #<navigatorclass @ #xcebfee> :Transit_to_task_location!
      NAV --> QMOW : Transit_to_task_location!

2.CO -->OOD: Task_location_reached?
   OOD --> #<navigatorclass @ #xcebfee> :Task_location_reached?
      NAV --> QMOW : Task_location_reached? y
OOD -->CO: yes

3.CO -->OOD: Critical_Systems_OK?
   OOD --> #<engineerclass @ #xcec8f6> :Critical_Systems_OK?
      ENG --> EOOW :  Critical_Systems_OK? y
OOD -->CO: yes

3.CO -->OOD: Search_area_for_target!
   OOD --> #<weapons-officerclass @ #xceb2de> :Search_area_for_target!
      WEPS --> SONAR: Search_area_for_target!

3.CO -->OOD: Search_pattern_completed?
   OOD --> #<weapons-officerclass @ #xceb2de> :Search_pattern_completed?
      WEPS --> SONAR: Search_pattern_completed? y
OOD -->CO: yes

31.CO -->OOD: Critical_Systems_OK?
   OOD --> #<engineerclass @ #xcec8f6> :Critical_Systems_OK?
      ENG --> EOOW :  Critical_Systems_OK? y
OOD -->CO: yes

31.CO -->OOD: Target_found?
   OOD --> #<weapons-officerclass @ #xceb2de> :Target_found?
      WEPS --> SONAR: Target_found? y
OOD -->CO: yes

4.CO -->OOD: Critical_Systems_OK?
   OOD --> #<engineerclass @ #xcec8f6> :Critical_Systems_OK?
      ENG --> EOOW :  Critical_Systems_OK? y
OOD -->CO: yes
```

```
4.CO -->OOD: Commence_task_on_target!
   OOD --> #<weapons-officerclass @ #xceb2de> :Commence_task_on_target!
      WEPS --> SONAR: Commence_task_on_target!

4.CO -->OOD: Standoff_distance_reached?
   OOD --> #<weapons-officerclass @ #xceb2de> :Standoff_distance_reached?
      WEPS --> SONAR: Standoff_distance_reached? y


4.CO -->OOD: Drop_package!
   OOD --> #<weapons-officerclass @ #xceb2de> :Drop_package!
      WEPS --> SONAR: Drop_package!

4.CO -->OOD: Is_package_dropped?
   OOD --> #<weapons-officerclass @ #xceb2de> :Is_package_dropped?
      WEPS --> SONAR: Is_package_dropped? y
OOD -->CO: yes

4.CO -->OOD: Get_GPS_fix!
   OOD --> #<navigatorclass @ #xcebfee> :Get_GPS_fix!
      NAV --> QMOW : Get_GPS_fix!

4.CO -->OOD: GPS_fix_obtained?
   OOD --> #<navigatorclass @ #xcebfee> :GPS_fix_obtained?
      NAV --> QMOW : GPS_fix_obtained? y
OOD -->CO: yes

5.CO -->OOD: Critical_Systems_OK?
   OOD --> #<engineerclass @ #xcec8f6> :Critical_Systems_OK?
      ENG --> EOOW :  Critical_Systems_OK? y
OOD -->CO: yes

5.CO -->OOD: Transit_to_task_location!
   OOD --> #<navigatorclass @ #xcebfee> :Transit_to_task_location!
      NAV --> QMOW : Transit_to_task_location!

5.CO -->OOD: Task_location_reached?
   OOD --> #<navigatorclass @ #xcebfee> :Task_location_reached?
      NAV --> QMOW : Task_location_reached? y
OOD -->CO: yes

6.CO -->OOD: Critical_Systems_OK?
```

```
    OOD --> #<engineerclass @ #xcec8f6> :Critical_Systems_OK?
        ENG --> EOOW :  Critical_Systems_OK? y
OOD -->CO: yes

6.CO -->OOD: Search_area_for_target!
    OOD --> #<weapons-officerclass @ #xceb2de> :Search_area_for_target!
        WEPS --> SONAR: Search_area_for_target!

6.CO -->OOD: Search_pattern_completed?
    OOD --> #<weapons-officerclass @ #xceb2de> :Search_pattern_completed?
        WEPS --> SONAR: Search_pattern_completed? y
OOD -->CO: yes

61.CO -->OOD: Critical_Systems_OK?
    OOD --> #<engineerclass @ #xcec8f6> :Critical_Systems_OK?
        ENG --> EOOW :  Critical_Systems_OK? y
OOD -->CO: yes


    OOD --> #<weapons-officerclass @ #xceb2de> :Target_found?
        WEPS --> SONAR: Target_found? y
OOD -->CO: yes

7.CO -->OOD: Critical_Systems_OK?
    OOD --> #<engineerclass @ #xcec8f6> :Critical_Systems_OK?
        ENG --> EOOW :  Critical_Systems_OK? y
OOD -->CO: yes

7.CO -->OOD: Commence_task_on_target!
    OOD --> #<weapons-officerclass @ #xceb2de> :Commence_task_on_target!
        WEPS --> SONAR: Commence_task_on_target!

7.CO -->OOD: Standoff_distance_reached?
    OOD --> #<weapons-officerclass @ #xceb2de> :Standoff_distance_reached?
        WEPS --> SONAR: Standoff_distance_reached? y
OOD -->CO: yes

7.CO -->OOD: Drop_package!
    OOD --> #<weapons-officerclass @ #xceb2de> :Drop_package!
        WEPS --> SONAR: Drop_package!

7.CO -->OOD: Is_package_dropped?
    OOD --> #<weapons-officerclass @ #xceb2de> :Is_package_dropped?
```

```
        WEPS --> SONAR: Is_package_dropped? y
OOD -->CO: yes

7.CO -->OOD: Get_GPS_fix!
    OOD --> #<navigatorclass @ #xcebfee> :Get_GPS_fix!
        NAV --> QMOW : Get_GPS_fix!

7.CO -->OOD: GPS_fix_obtained?
    OOD --> #<navigatorclass @ #xcebfee> :GPS_fix_obtained?
        NAV --> QMOW : GPS_fix_obtained? y
OOD -->CO: yes

8.CO -->OOD: Critical_Systems_OK?
    OOD --> #<engineerclass @ #xcec8f6> :Critical_Systems_OK?
        ENG --> EOOW :  Critical_Systems_OK? y
OOD -->CO: yes

8.CO -->OOD: Transit_to_recovery_point!
    OOD --> #<navigatorclass @ #xcebfee> :Transit_to_recovery_point!
        NAV --> QMOW : Transit_to_recovery_point!

8.CO -->OOD: At_recovery_point?
    OOD --> #<navigatorclass @ #xcebfee> :At_recovery_point?
        NAV --> QMOW : At_recovery_point? y
OOD -->CO: yes

mission_complete.CO -->OOD: Mission_complete!
    OOD --> #<engineerclass @ #xcec8f6> :Mission_complete!
```

2. LISP Trace - Failed Critical System Prior to Initialization

```
user(1): (load "ood.cl") (load "co.cl") (execute_mission)

; Loading /users/work2/mjholden/thesis/mythesis/LISP/ood.cl.
t
user(2):
; Loading /users/work2/mjholden/thesis/mythesis/LISP/co.cl.
t
user(3):

1.CO -->OOD: Critical_Systems_OK?
   OOD --> #<engineerclass @ #xcec8f6> :Critical_Systems_OK?
      ENG --> EOOW :  Critical_Systems_OK? n
OOD -->CO: no

mission_abort.CO -->OOD: Abort_mission!
   OOD --> #<navigatorclass @ #xcebfee> :Is_recovery_point_obtainable?
      NAV --> QMOW : Is_recovery_point_obtainable? n
   OOD --> #<engineerclass @ #xcec8f6> :Abort_mission!
      ENG --> EOOW : Abort_mission!
done
```

3. LISP Trace - Failed Initialization

```
user(1): (load "ood.cl") (load "co.cl") (execute_mission)

; Loading /users/work2/mjholden/thesis/mythesis/LISP/ood.cl.
t
user(2):
; Loading /users/work2/mjholden/thesis/mythesis/LISP/co.cl.
t
user(3):

1.CO -->OOD: Critical_Systems_OK?
   OOD --> #<engineerclass @ #xcec8f6> :Critical_Systems_OK?
      ENG --> EOOW :  Critical_Systems_OK? n
OOD -->CO: no

mission_abort.CO -->OOD: Abort_mission!
   OOD --> #<navigatorclass @ #xcebfee> :Is_recovery_point_obtainable?
      NAV --> QMOW : Is_recovery_point_obtainable? n
   OOD --> #<engineerclass @ #xcec8f6> :Abort_mission!
      ENG --> EOOW : Abort_mission!
done
user(4):  (load "ood.cl") (load "co.cl") (execute_mission)

; Loading /users/work2/mjholden/thesis/mythesis/LISP/ood.cl.
t
user(5):
; Loading /users/work2/mjholden/thesis/mythesis/LISP/co.cl.
t
user(6):

1.CO -->OOD: Critical_Systems_OK?
   OOD --> #<engineerclass @ #xcfdd46> :Critical_Systems_OK?
      ENG --> EOOW :  Critical_Systems_OK? y
OOD -->CO: yes

1.CO -->OOD: Initialize_vehicle!
   OOD --> #<engineerclass @ #xcfdd46> :Initialize_vehicle!
      ENG --> EOOW : Initialize_vehicle!
   OOD --> #<navigatorclass @ #xcfdcfe> :Initialize_vehicle!
      NAV --> QMOW : Initialize_vehicle!
   OOD --> #<weapons-officerclass @ #xcfdcb6> :Initialize_vehicle!
      WEPS --> SONAR: Initialize_vehicle!
```

```
1.CO -->OOD: Initialization_complete?
   OOD --> #<weapons-officerclass @ #xcfdcb6> :Initialization_complete?
      WEPS --> SONAR: Initialization_complete? n
OOD -->CO: no

1.CO -->OOD: Initialization_aborted?
   OOD --> #<weapons-officerclass @ #xcfdcb6> :Initialization_aborted?
      WEPS --> SONAR: Initialization_aborted? y
OOD -->CO: yes

mission_abort.CO -->OOD: Abort_mission!

   OOD --> #<navigatorclass @ #xcfdcfe> :Is_recovery_point_obtainable?
      NAV --> QMOW : Is_recovery_point_obtainable? n
   OOD --> #<engineerclass @ #xcfdd46> :Abort_mission!
      ENG --> EOOW : Abort_mission!
done
```

4. LISP Trace - Failed Navigation System

```
...
1.CO -->OOD: Initialization_complete?
   OOD --> #<weapons-officerclass @ #xde27ce> :Initialization_complete?
      WEPS --> SONAR: Initialization_complete? y
   OOD --> #<navigatorclass @ #xde2816> :Initialization_complete?
      NAV --> QMOW : Initialization_complete? y
   OOD --> #<engineerclass @ #xde285e> :Initialization_complete?
      ENG --> EOOW :  Initialization_complete? y
OOD -->CO: yes

2.CO -->OOD: Critical_Systems_OK?
   OOD --> #<engineerclass @ #xde285e> :Critical_Systems_OK?
      ENG --> EOOW :  Critical_Systems_OK? y
OOD -->CO: yes

2.CO -->OOD: Transit_to_task_location!
   OOD --> #<navigatorclass @ #xde2816> :Transit_to_task_location!
      NAV --> QMOW : Transit_to_task_location!

2.CO -->OOD: Task_location_reached?
   OOD --> #<navigatorclass @ #xde2816> :Task_location_reached?
      NAV --> QMOW : Task_location_reached? n
OOD -->CO: no

2.CO -->OOD: Waypoint_reached?
   OOD --> #<navigatorclass @ #xde2816> :Waypoint_reached?
      NAV --> QMOW : Waypoint_reached? n
OOD -->CO: no

2.CO -->OOD: Waypoint_process_OK?
   OOD --> #<navigatorclass @ #xde2816> :Waypoint_process_OK?
      NAV --> QMOW : Waypoint_process_OK? n
OOD -->CO: no

mission_abort.CO -->OOD: Abort_mission!
   OOD --> #<navigatorclass @ #xde2816> :Is_recovery_point_obtainable?
      NAV --> QMOW : Is_recovery_point_obtainable? n
   OOD --> #<engineerclass @ #xde285e> :Abort_mission!
      ENG --> EOOW : Abort_mission!
done
```

5. LISP Trace - Failed Search

```
...
2.CO -->OOD: Task_location_reached?
   OOD --> #<navigatorclass @ #xdf32ee> :Task_location_reached?
      NAV --> QMOW : Task_location_reached? y
OOD -->CO: yes

3.CO -->OOD: Critical_Systems_OK?
   OOD --> #<engineerclass @ #xdf3336> :Critical_Systems_OK?
      ENG --> EOOW :  Critical_Systems_OK? y
OOD -->CO: yes

3.CO -->OOD: Search_area_for_target!
   OOD --> #<weapons-officerclass @ #xdf32a6> :Search_area_for_target!
      WEPS --> SONAR: Search_area_for_target!

3.CO -->OOD: Search_pattern_completed?
   OOD --> #<weapons-officerclass @ #xdf32a6> :Search_pattern_completed?
      WEPS --> SONAR: Search_pattern_completed? n
OOD -->CO: no

3.CO -->OOD: Sonar_failure?
   OOD --> #<weapons-officerclass @ #xdf32a6> :Sonar_failure?
      WEPS --> SONAR: Sonar_failure? y
OOD -->CO: yes

mission_abort.CO -->OOD: Abort_mission!
   OOD --> #<navigatorclass @ #xdf32ee> :Is_recovery_point_obtainable?
      NAV --> QMOW : Is_recovery_point_obtainable? n
   OOD --> #<engineerclass @ #xdf3336> :Abort_mission!
      ENG --> EOOW : Abort_mission!
done
```

6. LISP Trace - No Targets in Search Area

```
...
3.CO -->OOD: Search_area_for_target!
   OOD --> #<weapons-officerclass @ #xe03976> :Search_area_for_target!
      WEPS --> SONAR: Search_area_for_target!

3.CO -->OOD: Search_pattern_completed?
   OOD --> #<weapons-officerclass @ #xe03976> :Search_pattern_completed?
      WEPS --> SONAR: Search_pattern_completed? y
OOD -->CO: yes

31.CO -->OOD: Critical_Systems_OK?
   OOD --> #<engineerclass @ #xe03a06> :Critical_Systems_OK?
      ENG --> EOOW :  Critical_Systems_OK? y
OOD -->CO: yes

31.CO -->OOD: Target_found?
   OOD --> #<weapons-officerclass @ #xe03976> :Target_found?
      WEPS --> SONAR: Target_found? n
OOD -->CO: no

5.CO -->OOD: Critical_Systems_OK?
   OOD --> #<engineerclass @ #xe03a06> :Critical_Systems_OK?
      ENG --> EOOW :  Critical_Systems_OK? y
OOD -->CO: yes

5.CO -->OOD: Transit_to_task_location!
   OOD --> #<navigatorclass @ #xe039be> :Transit_to_task_location!
      NAV --> QMOW : Transit_to_task_location!
...
```

7. LISP Trace - Failed Homing System

```
...
3.CO -->OOD: Search_pattern_completed?
   OOD --> #<weapons-officerclass @ #xce5f6e> :Search_pattern_completed?
      WEPS --> SONAR: Search_pattern_completed? y
OOD -->CO: yes

31.CO -->OOD: Critical_Systems_OK?
   OOD --> #<engineerclass @ #xce5ffe> :Critical_Systems_OK?
      ENG --> EOOW :  Critical_Systems_OK? y
OOD -->CO: yes

31.CO -->OOD: Target_found?
   OOD --> #<weapons-officerclass @ #xce5f6e> :Target_found?
      WEPS --> SONAR: Target_found? y
OOD -->CO: yes

4.CO -->OOD: Critical_Systems_OK?
   OOD --> #<engineerclass @ #xce5ffe> :Critical_Systems_OK?
      ENG --> EOOW :  Critical_Systems_OK? y
OOD -->CO: yes

4.CO -->OOD: Commence_task_on_target!
   OOD --> #<weapons-officerclass @ #xce5f6e> :Commence_task_on_target!
      WEPS --> SONAR: Commence_task_on_target!

4.CO -->OOD: Standoff_distance_reached?
   OOD --> #<weapons-officerclass @ #xce5f6e> :Standoff_distance_reached?
      WEPS --> SONAR: Standoff_distance_reached? n
OOD -->CO: no

4.CO -->OOD: Abort_homing?
   OOD --> #<weapons-officerclass @ #xce5f6e> :Abort_homing?
      WEPS --> SONAR: Abort_homing? y
OOD -->CO: yes

mission_abort.CO -->OOD: Abort_mission!
   OOD --> #<navigatorclass @ #xce5fb6> :Is_recovery_point_obtainable?
      NAV --> QMOW : Is_recovery_point_obtainable? n
   OOD --> #<engineerclass @ #xce5ffe> :Abort_mission!
      ENG --> EOOW : Abort_mission!
done
```

8. LISP Trace - Successful Task Completion

```
...
4.CO -->OOD: Critical_Systems_OK?
   OOD --> #<engineerclass @ #xcf7336> :Critical_Systems_OK?
      ENG --> EOOW :  Critical_Systems_OK? y
OOD -->CO: yes

4.CO -->OOD: Commence_task_on_target!
   OOD --> #<weapons-officerclass @ #xcf72a6> :Commence_task_on_target!
      WEPS --> SONAR: Commence_task_on_target!

4.CO -->OOD: Standoff_distance_reached?
   OOD --> #<weapons-officerclass @ #xcf72a6> :Standoff_distance_reached?
      WEPS --> SONAR: Standoff_distance_reached? y
OOD -->CO: yes

4.CO -->OOD: Drop_package!
   OOD --> #<weapons-officerclass @ #xcf72a6> :Drop_package!
      WEPS --> SONAR: Drop_package!

4.CO -->OOD: Is_package_dropped?
   OOD --> #<weapons-officerclass @ #xcf72a6> :Is_package_dropped?
      WEPS --> SONAR: Is_package_dropped? y
OOD -->CO: yes

4.CO -->OOD: Get_GPS_fix!
   OOD --> #<navigatorclass @ #xcf72ee> :Get_GPS_fix!
      NAV --> QMOW : Get_GPS_fix!

4.CO -->OOD: GPS_fix_obtained?
   OOD --> #<navigatorclass @ #xcf72ee> :GPS_fix_obtained?
      NAV --> QMOW : GPS_fix_obtained? y
OOD -->CO: yes

5.CO -->OOD: Critical_Systems_OK?
   OOD --> #<engineerclass @ #xcf7336> :Critical_Systems_OK?
      ENG --> EOOW :  Critical_Systems_OK? y
OOD -->CO: yes

5.CO -->OOD: Transit_to_task_location!

...
```

9. LISP Trace - Uncharted Obstacle Replanning

```
...
5.CO -->OOD: Waypoint_reached?
   OOD --> #<navigatorclass @ #xcf72ee> :Waypoint_reached?
      NAV --> QMOW : Waypoint_reached? n
OOD -->CO: no

5.CO -->OOD: Waypoint_process_OK?
   OOD --> #<navigatorclass @ #xcf72ee> :Waypoint_process_OK?
      NAV --> QMOW : Waypoint_process_OK? y
OOD -->CO: yes

5.CO -->OOD: NonCritical_Systems_OK?
   OOD --> #<weapons-officerclass @ #xcf72a6> :NonCritical_Systems_OK?
      WEPS --> SONAR: NonCritical_Systems_OK? y
   OOD --> #<engineerclass @ #xcf7336> :NonCritical_Systems_OK?
      ENG --> EOOW :  NonCritical_Systems_OK? y
OOD -->CO: yes

5.CO -->OOD: Area_clear_of_uncharted_obstacles?
   OOD --> #<navigatorclass @ #xcf72ee> :Area_clear_of_uncharted_obstacles?
      NAV --> QMOW : Area_clear_of_uncharted_obstacles? n
OOD -->CO: no

5.CO -->OOD: Log_new_obstacle!
   OOD --> #<navigatorclass @ #xcf72ee> :Log_new_obstacle!
      NAV --> QMOW : Log_new_obstacle!

5.CO -->OOD: New_obstacle_logged?
   OOD --> #<navigatorclass @ #xcf72ee> :New_obstacle_logged?
      NAV --> QMOW : New_obstacle_logged? y
OOD -->CO: yes

5.CO -->OOD: Loiter_and_Start_Local_Replanner!
   OOD --> #<navigatorclass @ #xcf72ee> :Loiter_and_Start_Local_Replanner!
      NAV --> QMOW : Loiter_and_Start_Local_Replanner!

5.CO -->OOD: Setpoints_and_modes_sent?
   OOD --> #<navigatorclass @ #xcf72ee> :Setpoints_and_modes_sent?
      NAV --> QMOW : Setpoints_and_modes_sent? y
OOD -->CO: yes

5.CO -->OOD: Critical_Systems_OK?
```

```
    OOD --> #<engineerclass @ #xcf7336> :Critical_Systems_OK?
        ENG --> EOOW :  Critical_Systems_OK? y
OOD -->CO: yes

5.CO -->OOD: Transit_to_task_location!

...
```

10. LISP Trace - Task Temporarily Incomplete but not Failed

```
...
7.CO -->OOD: Standoff_distance_reached?
   OOD --> #<weapons-officerclass @ #xcf72a6> :Standoff_distance_reached?
      WEPS --> SONAR: Standoff_distance_reached? n
OOD -->CO: no

7.CO -->OOD: Abort_homing?
   OOD --> #<weapons-officerclass @ #xcf72a6> :Abort_homing?
      WEPS --> SONAR: Abort_homing? n
OOD -->CO: no

7.CO -->OOD: Standoff_distance_reached?
   OOD --> #<weapons-officerclass @ #xcf72a6> :Standoff_distance_reached?
      WEPS --> SONAR: Standoff_distance_reached? y
OOD -->CO: yes

7.CO -->OOD: Drop_package!
   OOD --> #<weapons-officerclass @ #xcf72a6> :Drop_package!
      WEPS --> SONAR: Drop_package!

7.CO -->OOD: Is_package_dropped?
   OOD --> #<weapons-officerclass @ #xcf72a6> :Is_package_dropped?
      WEPS --> SONAR: Is_package_dropped? n
OOD -->CO: no

7.CO -->OOD: Is_package_drop_aborted?
   OOD --> #<weapons-officerclass @ #xcf72a6> :Is_package_drop_aborted?
      WEPS --> SONAR: Is_package_drop_aborted? n
OOD -->CO: no

7.CO -->OOD: Is_package_dropped?
   OOD --> #<weapons-officerclass @ #xcf72a6> :Is_package_dropped?
      WEPS --> SONAR: Is_package_dropped? n
OOD -->CO: no

7.CO -->OOD: Is_package_drop_aborted?
   OOD --> #<weapons-officerclass @ #xcf72a6> :Is_package_drop_aborted?
      WEPS --> SONAR: Is_package_drop_aborted? n
OOD -->CO: no

7.CO -->OOD: Is_package_dropped?
   OOD --> #<weapons-officerclass @ #xcf72a6> :Is_package_dropped?
```

```
        WEPS --> SONAR: Is_package_dropped? y
OOD -->CO: yes

7.CO -->OOD: Get_GPS_fix!
    OOD --> #<navigatorclass @ #xcf72ee> :Get_GPS_fix!
        NAV --> QMOW : Get_GPS_fix!

...
```

11. LISP Trace - Waypoint Not Yet Reached

```
...
2.CO -->OOD: Transit_to_task_location!
   OOD --> #<navigatorclass @ #xdea336> :Transit_to_task_location!
      NAV --> QMOW : Transit_to_task_location!

2.CO -->OOD: Task_location_reached?
   OOD --> #<navigatorclass @ #xdea336> :Task_location_reached?
      NAV --> QMOW : Task_location_reached? n
OOD -->CO: no

2.CO -->OOD: Waypoint_reached?
   OOD --> #<navigatorclass @ #xdea336> :Waypoint_reached?
      NAV --> QMOW : Waypoint_reached? n
OOD -->CO: no

2.CO -->OOD: Waypoint_process_OK?
   OOD --> #<navigatorclass @ #xdea336> :Waypoint_process_OK?
      NAV --> QMOW : Waypoint_process_OK? y
OOD -->CO: yes

2.CO -->OOD: NonCritical_Systems_OK?
   OOD --> #<weapons-officerclass @ #xdea2ee> :NonCritical_Systems_OK?
      WEPS --> SONAR: NonCritical_Systems_OK? y
   OOD --> #<engineerclass @ #xdea37e> :NonCritical_Systems_OK?
      ENG --> EOOW :  NonCritical_Systems_OK? y
OOD -->CO: yes

2.CO -->OOD: Area_clear_of_uncharted_obstacles?
   OOD --> #<navigatorclass @ #xdea336> :Area_clear_of_uncharted_obstacles?
      NAV --> QMOW : Area_clear_of_uncharted_obstacles? y
OOD -->CO: yes

2.CO -->OOD: Setpoints_and_modes_sent?
   OOD --> #<navigatorclass @ #xdea336> :Setpoints_and_modes_sent?
      NAV --> QMOW : Setpoints_and_modes_sent? y
OOD -->CO: yes

2.CO -->OOD: Critical_Systems_OK?
   OOD --> #<engineerclass @ #xdea37e> :Critical_Systems_OK?
      ENG --> EOOW :  Critical_Systems_OK? y
OOD -->CO: yes
2.CO -->OOD: Transit_to_task_location!
```

```
    OOD --> #<navigatorclass @ #xdea336> :Transit_to_task_location!
        NAV --> QMOW : Transit_to_task_location!

2.CO -->OOD: Task_location_reached?
    OOD --> #<navigatorclass @ #xdea336> :Task_location_reached?
        NAV --> QMOW : Task_location_reached? n
OOD -->CO: no

2.CO -->OOD: Waypoint_reached?
    OOD --> #<navigatorclass @ #xdea336> :Waypoint_reached?
        NAV --> QMOW : Waypoint_reached? y
OOD -->CO: yes

2.CO -->OOD: Get_next_waypoint!

2.CO -->OOD: NonCritical_Systems_OK?
    OOD --> #<weapons-officerclass @ #xdea2ee> :NonCritical_Systems_OK?
        WEPS --> SONAR: NonCritical_Systems_OK? y
    OOD --> #<engineerclass @ #xdea37e> :NonCritical_Systems_OK?
        ENG --> EOOW :  NonCritical_Systems_OK? y
OOD -->CO: yes

2.CO -->OOD: Area_clear_of_uncharted_obstacles?
    OOD --> #<navigatorclass @ #xdea336> :Area_clear_of_uncharted_obstacles?
        NAV --> QMOW : Area_clear_of_uncharted_obstacles? y
OOD -->CO: yes

2.CO -->OOD: Setpoints_and_modes_sent?
    OOD --> #<navigatorclass @ #xdea336> :Setpoints_and_modes_sent?
        NAV --> QMOW : Setpoints_and_modes_sent? y
OOD -->CO: yes

2.CO -->OOD: Critical_Systems_OK?
    OOD --> #<engineerclass @ #xdea37e> :Critical_Systems_OK?
        ENG --> EOOW :  Critical_Systems_OK? y
OOD -->CO: yes

2.CO -->OOD: Transit_to_task_location!
    OOD --> #<navigatorclass @ #xdea336> :Transit_to_task_location!
        NAV --> QMOW : Transit_to_task_location!

2.CO -->OOD: Task_location_reached?
    OOD --> #<navigatorclass @ #xdea336> :Task_location_reached?
```

```
          NAV --> QMOW : Task_location_reached? y
OOD -->CO: yes

3.CO -->OOD: Critical_Systems_OK?
     OOD --> #<engineerclass @ #xdea37e> :Critical_Systems_OK?
          ENG --> EOOW :  Critical_Systems_OK? y
OOD -->CO: yes

3.CO -->OOD: Search_area_for_target!

...
```

12. LISP Trace - Non Critical Systems Failure

```
...
2.CO -->OOD: NonCritical_Systems_OK?
   OOD --> #<weapons-officerclass @ #xdfda5e> :NonCritical_Systems_OK?
      WEPS --> SONAR: NonCritical_Systems_OK? y
   OOD --> #<engineerclass @ #xdfdaee> :NonCritical_Systems_OK?
      ENG --> EOOW :  NonCritical_Systems_OK? n
OOD -->CO: no

2.CO -->OOD: Loiter_and_Start_Global_Replanner!
   OOD --> #<navigatorclass @ #xdfdaa6> :Loiter_and_Start_Global_Replanner!
      NAV --> QMOW : Loiter_and_Start_Global_Replanner!

2.CO -->OOD: Setpoints_and_modes_sent?
   OOD --> #<navigatorclass @ #xdfdaa6> :Setpoints_and_modes_sent?
      NAV --> QMOW : Setpoints_and_modes_sent? y
OOD -->CO: yes

2.CO -->OOD: Critical_Systems_OK?
   OOD --> #<engineerclass @ #xdfdaee> :Critical_Systems_OK?
      ENG --> EOOW :  Critical_Systems_OK? y
OOD -->CO: yes

2.CO -->OOD: Transit_to_task_location!
   OOD --> #<navigatorclass @ #xdfdaa6> :Transit_to_task_location!
      NAV --> QMOW : Transit_to_task_location!

...
```

B. ADA TRACES

1. Ada Trace - Fully Successful Mission

```
> go
Enter mission file name =>  m1

 1. CO --> OOD: Get Mission Orders!
 1.   OOD --> CO: Get Mission Orders, Aye sir!

Phase     Action                    Next Phase to go to
-----------------------------------------------------------
  1    Initialize vehicle          If Succeed/Fail =   2/ 98
  2    Transit to task location    If Succeed/Fail =   3/  2
  3    Search area for target      If Succeed/Fail =   4/  3
  4    Report targets found        If Succeed/Fail =   5/  6
  5    Commence task on target     If Succeed/Fail =   6/  5
  6    Transit to task location    If Succeed/Fail =   7/  6
  7    Search area for target      If Succeed/Fail =   8/  7
  8    Report targets found        If Succeed/Fail =   9/ 10
  9    Commence task on target     If Succeed/Fail =  10/  9
 10     Transit to recovery point  If Succeed/Fail =  99/ 10
 11     Abort Mission(98)          If Succeed/Fail =  98/ 98
 12     Mission Complete(99)       If Succeed/Fail =  99/ 99
-----------------------------------------------------------

 1. CO --> OOD: Are Critical Systems OK?
 1.   OOD  --> ENG: Are Critical Systems OK, Eng? Answer => y
 1.   OOD --> CO: Critical Systems OK, Sir.
 1. CO --> OOD: Initialize!
 1.   OOD --> NAV, ENG, WEPS: Initialize!
 1.     NAV --> OOD: Initialize, Aye sir!
         NAV --> QMOW: Initialize!
          Initialize, QMOW, Aye sir!
 1.     ENG --> OOD: Initialize, Aye sir!
         ENG --> EOOW: Initialize
          Initialize, EOOW, Aye sir!
 1.     WEPS --> OOD: Initialize, Aye sir!
         WEPS --> TMOW: Initialize!
          Initialize, TMOW, Aye sir!
 1. CO --> OOD: Initialization Complete?
 1.   OOD  --> NAV: Initialization Complete, Nav? Answer => y
 1.   OOD  --> ENG: Initialization Complete, Eng? Answer => y
 1.   OOD  --> WEPS: Initialization Complete, Weps? Answer => y
```

```
1.  OOD --> CO: Initialization Complete, Sir.


2. CO --> OOD: Are Critical Systems OK?
2.  OOD --> ENG: Are Critical Systems OK, Eng? Answer => y
2.  OOD --> CO: Critical Systems OK, Sir.
2. CO --> OOD: Transit!
2.  OOD --> NAV: Transit!
2.    NAV --> OOD: Transit, Aye sir!
      NAV --> QMOW: Transit!
       Transit, QMOW, Aye sir!
2.  OOD --> NAV: Waypoint Reached, Nav? Answer => y

2.    NAV --> OOD: Get Next Waypoint, Aye sir!
      NAV --> QMOW: Get Next Waypoint!
       Get Next Waypoint, QMOW, Aye sir!
2.  OOD --> NAV: Got Next Waypoint, Nav? Answer => y
2.  OOD --> NAV: GPS Fix Needed, Nav? Answer => y
2.  OOD --> NAV: Get GPS Fix!
2.    NAV --> OOD: Get GPS Fix, Aye sir!
      NAV --> QMOW: Get GPS Fix!
       Get GPS Fix, QMOW, Aye sir!
2.  OOD --> NAV: GPS Fix Obtained, Nav? Answer => y
2.  OOD --> ENG: Are NonCritical Systems OK, Eng? Answer => y
2.  OOD --> NAV: Any Uncharted Obstacles, Nav? Answer => y
2.  OOD --> NAV: Commence Local Replanning!
2.    NAV --> OOD: Commence Local Replanning, Nav, Aye sir!
      NAV --> QMOW: Commence Local Replanning, Nav!
       Commence Local Replanning, Nav, QMOW, Aye sir!
2.  OOD --> NAV: Replanning Complete, Nav? Answer => y
2.  OOD --> NAV: Send Setpoints and Modes!
2.    NAV --> OOD: Send Setpoints and Modes, Nav, Aye sir!
      NAV --> QMOW: Send Setpoints and Modes, Nav!
       Send Setpoints and Modes, Nav, QMOW, Aye sir!
2.  OOD --> NAV: Setpoints sent, Nav? Answer => y
2. CO --> OOD: Transit complete?
2.  OOD --> NAV: Transit complete? Answer => y
2.  OOD --> CO: Transit Complete, Sir.


3. CO --> OOD: Are Critical Systems OK?
3.  OOD --> ENG: Are Critical Systems OK, Eng? Answer => y
3.  OOD --> CO: Critical Systems OK, Sir.
```

```
3. CO --> OOD: Search!
3.   OOD --> WEPS: Search!
3.     WEPS --> OOD: Search, Aye sir!
        WEPS --> TMOW: Search!
         Search, TMOW, Aye sir!
3.   OOD --> WEPS: Search Complete, Weps? Answer => y
3. CO --> OOD: Search Pattern Complete?
3.   OOD --> WEPS: Search Pattern Complete? Answer => y
3.   OOD  --> CO: Search Pattern Complete, Sir.


4. CO --> OOD: Are Critical Systems OK?
4.   OOD  --> ENG: Are Critical Systems OK, Eng? Answer => y
4.   OOD --> CO: Critical Systems OK, Sir.
4. CO --> OOD: Report targets found!
4.   OOD --> WEPS: Report targets found!
4.     WEPS --> OOD: Report targets found, Aye sir!
        WEPS --> TMOW: Report targets found!
         Report targets found, TMOW, Aye sir!
4. CO --> OOD: Any targets found?
4.   OOD --> WEPS: Any targets found? Answer => y
4.   OOD  --> CO: Targets in search area, Sir.


5. CO --> OOD: Are Critical Systems OK?
5.   OOD  --> ENG: Are Critical Systems OK, Eng? Answer => y
5.   OOD --> CO: Critical Systems OK, Sir.
5. CO --> OOD: Commence task on Target!
5.   OOD --> WEPS: Commence task on Target!
5.     WEPS --> OOD: Commence task on Target, Aye sir!
        WEPS --> TMOW: Commence task on Target!
         Commence task on Target, TMOW, Aye sir!
5.   OOD --> WEPS: Standoff Distance Reached, Weps? Answer => y
5.   OOD --> WEPS: Drop Package!
5.     WEPS --> OOD: Drop Package, Aye sir!
        WEPS --> TMOW: Drop Package!
         Drop Package, TMOW, Aye sir!
5.   OOD --> WEPS: Package Drop Complete, Weps? Answer => y
5.   OOD --> NAV: GPS Fix Needed, Nav? Answer => y
5.   OOD --> NAV: Get GPS Fix!
5.     NAV --> OOD: Get GPS Fix, Aye sir!
        NAV --> QMOW: Get GPS Fix!
         Get GPS Fix, QMOW, Aye sir!
```

```
5.  OOD --> NAV: GPS Fix Obtained, Nav? Answer => y
5. CO --> OOD: Task on Target Complete?
5.  OOD --> WEPS: Task on Target Complete? Answer => y
5.  OOD  --> CO: Task Complete, Sir.


6. CO --> OOD: Are Critical Systems OK?
6.  OOD  --> ENG: Are Critical Systems OK, Eng? Answer => y
6.  OOD --> CO: Critical Systems OK, Sir.
6. CO --> OOD: Transit!
6.  OOD --> NAV: Transit!
6.    NAV --> OOD: Transit, Aye sir!
      NAV --> QMOW: Transit!
       Transit, QMOW, Aye sir!
6.  OOD --> NAV: Waypoint Reached, Nav? Answer => y
6.  OOD --> NAV: Get Next Waypoint!
6.    NAV --> OOD: Get Next Waypoint, Aye sir!
      NAV --> QMOW: Get Next Waypoint!
       Get Next Waypoint, QMOW, Aye sir!
6.  OOD --> NAV: Got Next Waypoint, Nav? Answer => y
6.  OOD --> NAV: GPS Fix Needed, Nav? Answer => y
6.  OOD --> NAV: Get GPS Fix!
6.    NAV --> OOD: Get GPS Fix, Aye sir!
      NAV --> QMOW: Get GPS Fix!
       Get GPS Fix, QMOW, Aye sir!
6.  OOD --> NAV: GPS Fix Obtained, Nav? Answer => y
6.  OOD --> ENG: Are NonCritical Systems OK, Eng? Answer => y
6.  OOD --> NAV: Any Uncharted Obstacles, Nav? Answer => y
6.  OOD --> NAV: Commence Local Replanning!
6.    NAV --> OOD: Commence Local Replanning, Nav, Aye sir!
      NAV --> QMOW: Commence Local Replanning, Nav!
       Commence Local Replanning, Nav, QMOW, Aye sir!

6.  OOD --> NAV: Send Setpoints and Modes!
6.    NAV --> OOD: Send Setpoints and Modes, Nav, Aye sir!
      NAV --> QMOW: Send Setpoints and Modes, Nav!
       Send Setpoints and Modes, Nav, QMOW, Aye sir!
6.  OOD --> NAV: Setpoints sent, Nav? Answer => y
6. CO --> OOD: Transit complete?
6.  OOD  --> NAV: Transit complete? Answer => y
6.  OOD  --> CO: Transit Complete, Sir.
```

7. CO --> OOD: Are Critical Systems OK?
7. OOD --> ENG: Are Critical Systems OK, Eng? Answer => y
7. OOD --> CO: Critical Systems OK, Sir.
7. CO --> OOD: Search!
7. OOD --> WEPS: Search!
7. WEPS --> OOD: Search, Aye sir!
WEPS --> TMOW: Search!
Search, TMOW, Aye sir!
7. OOD --> WEPS: Search Complete, Weps? Answer => y
7. CO --> OOD: Search Pattern Complete?
7. OOD --> WEPS: Search Pattern Complete? Answer => y
7. OOD --> CO: Search Pattern Complete, Sir.

8. CO --> OOD: Are Critical Systems OK?
8. OOD --> ENG: Are Critical Systems OK, Eng? Answer => y
8. OOD --> CO: Critical Systems OK, Sir.
8. CO --> OOD: Report targets found!
8. OOD --> WEPS: Report targets found!
8. WEPS --> OOD: Report targets found, Aye sir!
WEPS --> TMOW: Report targets found!
Report targets found, TMOW, Aye sir!
8. CO --> OOD: Any targets found?
8. OOD --> WEPS: Any targets found? Answer => y
8. OOD --> CO: Targets in search area, Sir.

9. CO --> OOD: Are Critical Systems OK?
9. OOD --> ENG: Are Critical Systems OK, Eng? Answer => y
9. OOD --> CO: Critical Systems OK, Sir.
9. CO --> OOD: Commence task on Target!
9. OOD --> WEPS: Commence task on Target!
9. WEPS --> OOD: Commence task on Target, Aye sir!
WEPS --> TMOW: Commence task on Target!
Commence task on Target, TMOW, Aye sir!
9. OOD --> WEPS: Standoff Distance Reached, Weps? Answer => y
9. OOD --> WEPS: Drop Package!
9. WEPS --> OOD: Drop Package, Aye sir!
WEPS --> TMOW: Drop Package!
Drop Package, TMOW, Aye sir!
9. OOD --> WEPS: Package Drop Complete, Weps? Answer => y
9. OOD --> NAV: GPS Fix Needed, Nav? Answer => y

9. NAV --> OOD: Get GPS Fix, Aye sir!
NAV --> QMOW: Get GPS Fix!
Get GPS Fix, QMOW, Aye sir!
9. OOD --> NAV: GPS Fix Obtained, Nav? Answer => y
9. CO --> OOD: Task on Target Complete?
9. OOD --> WEPS: Task on Target Complete? Answer => y
9. OOD --> CO: Task Complete, Sir.

10. CO --> OOD: Are Critical Systems OK?
10. OOD --> ENG: Are Critical Systems OK, Eng? Answer => y
10. OOD --> CO: Critical Systems OK, Sir.
10. CO --> OOD: Transit to Recovery Point!
10. OOD --> NAV: Transit to Recovery Point!
10. NAV --> OOD: Transit to Recovery Point, Aye sir!
NAV --> QMOW: Transit to Recovery Point!
Transit to Recovery Point, QMOW, Aye sir!
10. CO --> OOD: At Recovery Point?
10. OOD --> NAV: At Recovery Point? Answer => y
10. OOD --> CO: Transit Complete, Sir.

CO --> OOD: Mission Complete!

2. Ada Trace - Failed Critical System Prior to Initialization

```
> go

Enter mission file name =>  m1

 1. CO --> OOD: Get Mission Orders!
 1.   OOD --> CO: Get Mission Orders, Aye sir!

Phase    Action                    Next Phase to go to
------------------------------------------------------------
 1    Initialize vehicle          If Succeed/Fail =  2/ 98
 2    Transit to task location    If Succeed/Fail =  3/  2
 3    Search area for target      If Succeed/Fail =  4/  3
 4    Report targets found        If Succeed/Fail =  5/  6
 5    Commence task on target     If Succeed/Fail =  6/  5
 6    Transit to task location    If Succeed/Fail =  7/  6
 7    Search area for target      If Succeed/Fail =  8/  7
 8    Report targets found        If Succeed/Fail =  9/ 10
 9    Commence task on target     If Succeed/Fail = 10/  9
10     Transit to recovery point  If Succeed/Fail = 99/ 10
11     Abort Mission(98)          If Succeed/Fail = 98/ 98
12     Mission Complete(99)       If Succeed/Fail = 99/ 99
------------------------------------------------------------

 1. CO --> OOD: Are Critical Systems OK?
 1.   OOD  --> ENG: Are Critical Systems OK, Eng? Answer => n
 1.   OOD --> CO: Critical Systems Failure, Sir.

    CO --> OOD: Mission Aborted!
```

3. Ada Trace - Failed Initialization

```
> go
Enter mission file name =>  m1

 1. CO --> OOD: Get Mission Orders!
 1.   OOD --> CO: Get Mission Orders, Aye sir!

Phase    Action                    Next Phase to go to
----------------------------------------------------------
 1    Initialize vehicle          If Succeed/Fail =  2/ 98
 2    Transit to task location    If Succeed/Fail =  3/  2
 3    Search area for target      If Succeed/Fail =  4/  3
 4    Report targets found        If Succeed/Fail =  5/  6
 5    Commence task on target     If Succeed/Fail =  6/  5
 6    Transit to task location    If Succeed/Fail =  7/  6
 7    Search area for target      If Succeed/Fail =  8/  7
 8    Report targets found        If Succeed/Fail =  9/ 10
 9    Commence task on target     If Succeed/Fail = 10/  9
10     Transit to recovery point  If Succeed/Fail = 99/ 10
11     Abort Mission(98)          If Succeed/Fail = 98/ 98
12     Mission Complete(99)       If Succeed/Fail = 99/ 99
----------------------------------------------------------
 1. CO --> OOD: Are Critical Systems OK?
 1.   OOD  --> ENG: Are Critical Systems OK, Eng? Answer => y
 1.   OOD --> CO: Critical Systems OK, Sir.
 1. CO --> OOD: Initialize!
 1.   OOD --> NAV, ENG, WEPS: Initialize!
 1.     NAV --> OOD: Initialize, Aye sir!
         NAV --> QMOW: Initialize!
          Initialize, QMOW, Aye sir!
 1.     ENG --> OOD: Initialize, Aye sir!
         ENG --> EOOW: Initialize
          Initialize, EOOW, Aye sir!
 1.     WEPS --> OOD: Initialize, Aye sir!
         WEPS --> TMOW: Initialize!
          Initialize, TMOW, Aye sir!
 1. CO --> OOD: Initialization Complete?
 1.   OOD  --> NAV: Initialization Complete, Nav? Answer => n
 1.   OOD  --> ENG: Initialization Complete, Eng? Answer => y
 1.   OOD  --> WEPS: Initialization Complete, Weps? Answer => y
 1.   OOD  --> CO: Initialization failure, Sir.

   CO --> OOD: Mission Aborted!
```

4. Ada Trace - Failed Navigation System

```
...
 1.  OOD  --> CO: Initialization Complete, Sir.


 2. CO --> OOD: Are Critical Systems OK?
 2.   OOD  --> ENG: Are Critical Systems OK, Eng? Answer => y
 2.   OOD --> CO: Critical Systems OK, Sir.
 2. CO --> OOD: Transit!
 2.   OOD --> NAV: Transit!
 2.     NAV --> OOD: Transit, Aye sir!
         NAV --> QMOW: Transit!
          Transit, QMOW, Aye sir!
 2.   OOD --> NAV: Waypoint Reached, Nav? Answer => n
 2.   OOD --> ENG: Are NonCritical Systems OK, Eng? Answer => n
 2.   OOD --> NAV: Any Uncharted Obstacles, Nav? Answer => n
 2.   OOD --> NAV: Commence Global Replanning!
 2.     NAV --> OOD: Commence Global Replanning, Nav, Aye sir!
         NAV --> QMOW: Commence Global Replanning, Nav!
          Commence Global Replanning, Nav, QMOW, Aye sir!
 2.   OOD --> NAV: Replanning Complete, Nav? Answer => y
 2.   OOD --> NAV: Send Setpoints and Modes!
 2.     NAV --> OOD: Send Setpoints and Modes, Nav, Aye sir!
         NAV --> QMOW: Send Setpoints and Modes, Nav!
          Send Setpoints and Modes, Nav, QMOW, Aye sir!
 2.   OOD --> NAV: Setpoints sent, Nav? Answer => n
 2.   OOD --> NAV: Navigation System Failure, Nav? Answer => y
 2.   OOD --> CO: Navigation Systems Failure, Sir.
 2. CO --> OOD: Transit complete?
 2.   OOD  --> NAV: Transit complete? Answer => n
 2.   OOD  --> CO: Transit incomplete, Sir.


    CO --> OOD: Mission Aborted!
```

5. Ada Trace - Failed Search

```
...
  2.   OOD  --> CO: Transit Complete, Sir.


 3.  CO --> OOD: Are Critical Systems OK?
 3.    OOD  --> ENG: Are Critical Systems OK, Eng? Answer => y
 3.    OOD --> CO: Critical Systems OK, Sir.
 3.  CO --> OOD: Search!
 3.    OOD --> WEPS: Search!
 3.      WEPS --> OOD: Search, Aye sir!
          WEPS --> TMOW: Search!
           Search, TMOW, Aye sir!
 3.    OOD --> WEPS: Search Complete, Weps? Answer => n
 3.    OOD --> WEPS: Abort Search, Weps? Answer => y
 3.    OOD --> CO: Search System Failure, Sir.
 3.  CO --> OOD: Search Pattern Complete?
 3.    OOD --> WEPS: Search Pattern Complete? Answer => n
 3.    OOD  --> CO: Search Pattern incomplete, Sir.


    CO --> OOD: Mission Aborted!
```

6. Ada Trace - No Targets in Search Area

```
...
 3.  OOD  --> CO: Search Pattern Complete, Sir.

 4. CO --> OOD: Are Critical Systems OK?
 4.  OOD  --> ENG: Are Critical Systems OK, Eng? Answer => y
 4.  OOD --> CO: Critical Systems OK, Sir.
 4. CO --> OOD: Report targets found!
 4.  OOD --> WEPS: Report targets found!
 4.    WEPS --> OOD: Report targets found, Aye sir!
       WEPS --> TMOW: Report targets found!
        Report targets found, TMOW, Aye sir!
 4. CO --> OOD: Any targets found?
 4.  OOD --> WEPS: Any targets found? Answer => n
 4.  OOD  --> CO: No targets in search area, Sir.

 6. CO --> OOD: Are Critical Systems OK?
 6.  OOD  --> ENG: Are Critical Systems OK, Eng? Answer => y
 6.  OOD --> CO: Critical Systems OK, Sir.
 6. CO --> OOD: Transit!
 6.  OOD --> NAV: Transit!
 6.    NAV --> OOD: Transit, Aye sir!
       NAV --> QMOW: Transit!
        Transit, QMOW, Aye sir!
 6.  OOD --> NAV: Waypoint Reached, Nav? Answer => y
 6.  OOD --> NAV: Get Next Waypoint!
 6.    NAV --> OOD: Get Next Waypoint, Aye sir!
       NAV --> QMOW: Get Next Waypoint!
        Get Next Waypoint, QMOW, Aye sir!
 6.  OOD --> NAV: Got Next Waypoint, Nav? Answer => y
 6.  OOD --> NAV: GPS Fix Needed, Nav? Answer => n
 6.  OOD --> ENG: Are NonCritical Systems OK, Eng? Answer => y
 6.  OOD --> NAV: Any Uncharted Obstacles, Nav? Answer => n
 6.  OOD --> NAV: Send Setpoints and Modes!
 6.    NAV --> OOD: Send Setpoints and Modes, Nav, Aye sir!
       NAV --> QMOW: Send Setpoints and Modes, Nav!
        Send Setpoints and Modes, Nav, QMOW, Aye sir!
 6.  OOD --> NAV: Setpoints sent, Nav? Answer => y
 6. CO --> OOD: Transit complete?
 6.  OOD  --> NAV: Transit complete? Answer => y
 6.  OOD  --> CO: Transit Complete, Sir.
...
```

7. Ada Trace - Failed Homing System

```
...
 8. CO --> OOD: Any targets found?
 8.   OOD --> WEPS: Any targets found? Answer => y
 8.   OOD  --> CO: Targets in search area, Sir.


 9. CO --> OOD: Are Critical Systems OK?
 9.   OOD  --> ENG: Are Critical Systems OK, Eng? Answer => y
 9.   OOD --> CO: Critical Systems OK, Sir.
 9. CO --> OOD: Commence task on Target!
 9.   OOD --> WEPS: Commence task on Target!
 9.     WEPS --> OOD: Commence task on Target, Aye sir!
         WEPS --> TMOW: Commence task on Target!
          Commence task on Target, TMOW, Aye sir!
 9.   OOD --> WEPS: Standoff Distance Reached, Weps? Answer => n
 9.   OOD --> WEPS: Abort Homing, Weps? Answer => y
 9.   OOD --> CO: Homing System Failure, Sir.
 9.   OOD --> NAV: GPS Fix Needed, Nav? Answer => n
 9. CO --> OOD: Task on Target Complete?
 9.   OOD --> WEPS: Task on Target Complete? Answer => n
 9.   OOD  --> CO: Task incomplete, Sir.


    CO --> OOD: Mission Aborted!
```

8. Ada Trace - Successful Task Completion

```
...
 3. CO --> OOD: Search Pattern Complete?
 3.   OOD --> WEPS: Search Pattern Complete? Answer => y
 3.   OOD  --> CO: Search Pattern Complete, Sir.

 4. CO --> OOD: Are Critical Systems OK?
 4.   OOD  --> ENG: Are Critical Systems OK, Eng? Answer => y
 4.   OOD --> CO: Critical Systems OK, Sir.
 4. CO --> OOD: Report targets found!
 4.   OOD --> WEPS: Report targets found!
 4.     WEPS --> OOD: Report targets found, Aye sir!
         WEPS --> TMOW: Report targets found!
          Report targets found, TMOW, Aye sir!
 4. CO --> OOD: Any targets found?
 4.   OOD --> WEPS: Any targets found? Answer => y
 4.   OOD  --> CO: Targets in search area, Sir.


 5. CO --> OOD: Are Critical Systems OK?
 5.   OOD  --> ENG: Are Critical Systems OK, Eng? Answer => y
 5.   OOD --> CO: Critical Systems OK, Sir.
 5. CO --> OOD: Commence task on Target!
 5.   OOD --> WEPS: Commence task on Target!
 5.     WEPS --> OOD: Commence task on Target, Aye sir!
         WEPS --> TMOW: Commence task on Target!
          Commence task on Target, TMOW, Aye sir!
 5.   OOD --> WEPS: Standoff Distance Reached, Weps? Answer => y
 5.   OOD --> WEPS: Drop Package!
 5.     WEPS --> OOD: Drop Package, Aye sir!
         WEPS --> TMOW: Drop Package!
          Drop Package, TMOW, Aye sir!
 5.   OOD --> WEPS: Package Drop Complete, Weps? Answer => y
 5.   OOD --> NAV: GPS Fix Needed, Nav? Answer => y
 5.   OOD --> NAV: Get GPS Fix!
 5.     NAV --> OOD: Get GPS Fix, Aye sir!
         NAV --> QMOW: Get GPS Fix!
          Get GPS Fix, QMOW, Aye sir!
 5.   OOD --> NAV: GPS Fix Obtained, Nav? Answer => y
 5. CO --> OOD: Task on Target Complete?
 5.   OOD --> WEPS: Task on Target Complete? Answer => y
 5.   OOD  --> CO: Task Complete, Sir.
...
```

9. Ada Trace - Uncharted Obstacle Replanning

```
...
 6. CO --> OOD: Are Critical Systems OK?
 6.   OOD  --> ENG: Are Critical Systems OK, Eng? Answer => y
 6.   OOD --> CO: Critical Systems OK, Sir.
 6. CO --> OOD: Transit!
 6.   OOD --> NAV: Transit!
 6.     NAV --> OOD: Transit, Aye sir!
         NAV --> QMOW: Transit!
          Transit, QMOW, Aye sir!
 6.   OOD --> NAV: Waypoint Reached, Nav? Answer => y
 6.   OOD --> NAV: Get Next Waypoint!
 6.     NAV --> OOD: Get Next Waypoint, Aye sir!
         NAV --> QMOW: Get Next Waypoint!
          Get Next Waypoint, QMOW, Aye sir!
 6.   OOD --> NAV: Got Next Waypoint, Nav? Answer => y
 6.   OOD --> NAV: GPS Fix Needed, Nav? Answer => y
 6.   OOD --> NAV: Get GPS Fix!
 6.     NAV --> OOD: Get GPS Fix, Aye sir!
         NAV --> QMOW: Get GPS Fix!
          Get GPS Fix, QMOW, Aye sir!
 6.   OOD --> NAV: GPS Fix Obtained, Nav? Answer => y
 6.   OOD --> ENG: Are NonCritical Systems OK, Eng? Answer => y
 6.   OOD --> NAV: Any Uncharted Obstacles, Nav? Answer => y
 6.   OOD --> NAV: Commence Local Replanning!
 6.     NAV --> OOD: Commence Local Replanning, Nav, Aye sir!
         NAV --> QMOW: Commence Local Replanning, Nav!
          Commence Local Replanning, Nav, QMOW, Aye sir!
 6.   OOD --> NAV: Replanning Complete, Nav? Answer => y
 6.   OOD --> NAV: Send Setpoints and Modes!
 6.     NAV --> OOD: Send Setpoints and Modes, Nav, Aye sir!
         NAV --> QMOW: Send Setpoints and Modes, Nav!
          Send Setpoints and Modes, Nav, QMOW, Aye sir!
 6.   OOD --> NAV: Setpoints sent, Nav? Answer => y
 6. CO --> OOD: Transit complete?
 6.   OOD  --> NAV: Transit complete? Answer => y
 6.   OOD  --> CO: Transit Complete, Sir.
...
```

10. Ada Trace - Task Temporarily Incomplete but not Failed

```
...
 8.  CO --> OOD: Any targets found?
 8.   OOD --> WEPS: Any targets found? Answer => y
 8.   OOD  --> CO: Targets in search area, Sir.


 9.  CO --> OOD: Are Critical Systems OK?
 9.   OOD  --> ENG: Are Critical Systems OK, Eng? Answer => y
 9.   OOD --> CO: Critical Systems OK, Sir.
 9.  CO --> OOD: Commence task on Target!
 9.   OOD --> WEPS: Commence task on Target!
 9.     WEPS --> OOD: Commence task on Target, Aye sir!
        WEPS --> TMOW: Commence task on Target!
         Commence task on Target, TMOW, Aye sir!
 9.   OOD --> WEPS: Standoff Distance Reached, Weps? Answer => n
 9.   OOD --> WEPS: Abort Homing, Weps? Answer => n
 9.   OOD --> NAV: GPS Fix Needed, Nav? Answer => n
 9.  CO --> OOD: Task on Target Complete?
 9.   OOD --> WEPS: Task on Target Complete? Answer => n
 9.   OOD  --> CO: Task incomplete, Sir.


 9.  CO --> OOD: Are Critical Systems OK?
 9.   OOD  --> ENG: Are Critical Systems OK, Eng? Answer => y
 9.   OOD --> CO: Critical Systems OK, Sir.
 9.  CO --> OOD: Commence task on Target!
 9.   OOD --> WEPS: Commence task on Target!
 9.     WEPS --> OOD: Commence task on Target, Aye sir!
        WEPS --> TMOW: Commence task on Target!
         Commence task on Target, TMOW, Aye sir!
 9.   OOD --> WEPS: Standoff Distance Reached, Weps? Answer => y
 9.   OOD --> WEPS: Drop Package!
 9.     WEPS --> OOD: Drop Package, Aye sir!
        WEPS --> TMOW: Drop Package!
         Drop Package, TMOW, Aye sir!
 9.   OOD --> WEPS: Package Drop Complete, Weps? Answer => n
 9.   OOD --> WEPS: Abort Drop, Weps? Answer => n
 9.   OOD --> NAV: GPS Fix Needed, Nav? Answer => n
 9.  CO --> OOD: Task on Target Complete?
 9.   OOD --> WEPS: Task on Target Complete? Answer => n
 9.   OOD  --> CO: Task incomplete, Sir.
```

```
9.  CO --> OOD: Are Critical Systems OK?
9.   OOD  --> ENG: Are Critical Systems OK, Eng? Answer => y
9.   OOD --> CO: Critical Systems OK, Sir.
9.  CO --> OOD: Commence task on Target!
9.   OOD --> WEPS: Commence task on Target!
9.     WEPS --> OOD: Commence task on Target, Aye sir!
       WEPS --> TMOW: Commence task on Target!
        Commence task on Target, TMOW, Aye sir!
9.   OOD --> WEPS: Standoff Distance Reached, Weps? Answer => y
9.   OOD --> WEPS: Drop Package!
9.     WEPS --> OOD: Drop Package, Aye sir!
       WEPS --> TMOW: Drop Package!
        Drop Package, TMOW, Aye sir!
9.   OOD --> WEPS: Package Drop Complete, Weps? Answer => y
9.   OOD --> NAV: GPS Fix Needed, Nav? Answer => n
9.  CO --> OOD: Task on Target Complete?
9.   OOD --> WEPS: Task on Target Complete? Answer => y
9.   OOD  --> CO: Task Complete, Sir.
...
```

11. Ada Trace - Waypoint Not Yet Reached

```
...
 1.   OOD  --> CO: Initialization Complete, Sir.


 2.  CO --> OOD: Are Critical Systems OK?
 2.   OOD  --> ENG: Are Critical Systems OK, Eng? Answer => y
 2.   OOD --> CO: Critical Systems OK, Sir.
 2.  CO --> OOD: Transit!
 2.   OOD --> NAV: Transit!
 2.     NAV --> OOD: Transit, Aye sir!
         NAV --> QMOW: Transit!
          Transit, QMOW, Aye sir!
 2.   OOD --> NAV: Waypoint Reached, Nav? Answer => n
 2.   OOD --> ENG: Are NonCritical Systems OK, Eng? Answer => y
 2.   OOD --> NAV: Any Uncharted Obstacles, Nav? Answer => n
 2.   OOD --> NAV: Send Setpoints and Modes!
 2.     NAV --> OOD: Send Setpoints and Modes, Nav, Aye sir!
         NAV --> QMOW: Send Setpoints and Modes, Nav!
          Send Setpoints and Modes, Nav, QMOW, Aye sir!
 2.   OOD --> NAV: Setpoints sent, Nav? Answer => y
 2.  CO --> OOD: Transit complete?
 2.   OOD  --> NAV: Transit complete? Answer => n
 2.   OOD  --> CO: Transit incomplete, Sir.


 2.  CO --> OOD: Are Critical Systems OK?
 2.   OOD  --> ENG: Are Critical Systems OK, Eng? Answer => y
 2.   OOD --> CO: Critical Systems OK, Sir.
 2.  CO --> OOD: Transit!
 2.   OOD --> NAV: Transit!
 2.     NAV --> OOD: Transit, Aye sir!
         NAV --> QMOW: Transit!
          Transit, QMOW, Aye sir!
 2.   OOD --> NAV: Waypoint Reached, Nav? Answer => y
 2.   OOD --> NAV: Get Next Waypoint!
 2.     NAV --> OOD: Get Next Waypoint, Aye sir!
         NAV --> QMOW: Get Next Waypoint!
          Get Next Waypoint, QMOW, Aye sir!
 2.   OOD --> NAV: Got Next Waypoint, Nav? Answer => y
 2.   OOD --> NAV: GPS Fix Needed, Nav? Answer => n
 2.   OOD --> ENG: Are NonCritical Systems OK, Eng? Answer => y
 2.   OOD --> NAV: Any Uncharted Obstacles, Nav? Answer => n
```

```
2.   OOD --> NAV: Send Setpoints and Modes!
2.     NAV --> OOD: Send Setpoints and Modes, Nav, Aye sir!
        NAV --> QMOW: Send Setpoints and Modes, Nav!
         Send Setpoints and Modes, Nav, QMOW, Aye sir!
2.   OOD --> NAV: Setpoints sent, Nav? Answer => y
2.  CO --> OOD: Transit complete?
2.   OOD  --> NAV: Transit complete? Answer => y
2.   OOD  --> CO: Transit Complete, Sir.
...
```

12. Ada Trace - Non Critical Systems Failure

```
...
 1.   OOD  --> CO: Initialization Complete, Sir.


 2.  CO --> OOD: Are Critical Systems OK?
 2.   OOD  --> ENG: Are Critical Systems OK, Eng? Answer => y
 2.   OOD --> CO: Critical Systems OK, Sir.
 2.  CO --> OOD: Transit!
 2.   OOD --> NAV: Transit!
 2.     NAV --> OOD: Transit, Aye sir!
        NAV --> QMOW: Transit!
         Transit, QMOW, Aye sir!
 2.   OOD --> NAV: Waypoint Reached, Nav? Answer => y
 2.   OOD --> NAV: Get Next Waypoint!
 2.     NAV --> OOD: Get Next Waypoint, Aye sir!
        NAV --> QMOW: Get Next Waypoint!
         Get Next Waypoint, QMOW, Aye sir!
 2.   OOD --> NAV: Got Next Waypoint, Nav? Answer => y
 2.   OOD --> NAV: GPS Fix Needed, Nav? Answer => n
 2.   OOD --> ENG: Are NonCritical Systems OK, Eng? Answer => n
 2.   OOD --> NAV: Any Uncharted Obstacles, Nav? Answer => n
 2.   OOD --> NAV: Commence Global Replanning!
 2.     NAV --> OOD: Commence Global Replanning, Nav, Aye sir!
        NAV --> QMOW: Commence Global Replanning, Nav!
         Commence Global Replanning, Nav, QMOW, Aye sir!
 2.   OOD --> NAV: Replanning Complete, Nav? Answer => y
 2.   OOD --> NAV: Send Setpoints and Modes!
 2.     NAV --> OOD: Send Setpoints and Modes, Nav, Aye sir!
        NAV --> QMOW: Send Setpoints and Modes, Nav!
         Send Setpoints and Modes, Nav, QMOW, Aye sir!
 2.   OOD --> NAV: Setpoints sent, Nav? Answer => y
 2.  CO --> OOD: Transit complete?
 2.   OOD  --> NAV: Transit complete? Answer => y
 2.   OOD  --> CO: Transit Complete, Sir.
...
```

LIST OF REFERENCES

1. Autonomous Robot Vehicles, I. J. Cox and G. T. Wilfong, eds., Springer-Verlag, New York, 1990.

2. Daily Reports on the World-Wide Web on the status of the Dante II vehicle during the Mount Spurr mission, http://maas-neotek.arc.nasa.gov/Dante/mission-status.html#status, 13 July - 19 August 1994.

3. Booch, G., *Software Engineering with Ada*, 3rd edition, Benjamin/Cummings, Redwood City, California, 1994.

4. Sha, Lui, and J.B. Goodenough, *Real-Time Scheduling Theory and Ada*, IEEE Computer, Volume 23, Number 4 , pp.53-62, April 1990.

5. Steele, R.D., and P.G. Backes, *Ada and real-time robotics: lessons learned,* IEEE Computer, Volume 27, Number 4, pp. 49-54, April 1994.

6. Howle, W.T., *Ada in Real-Time Embedded Systems: Orbital Maneuvering Vehicle(OMV)*, Proceedings of TRI-Ada '88, pp. 363-370, Charleston, WV, 24-27 October 1988.

7. Yen, M. *LISP-to-Ada Reenginineering issues and support Environments for fielding real-time systems*, Proceedings of the IEEE/AIAA 11th Digital Avionics Systems Conference, pp. 225-230, October 1992.

8. Healey, A. J., McGhee, R.B., Cristi, R.,Papoulias, F.A, Kwak, S.H., Kanayama, Y., Lee, M. J., Shukla, Zaky, E., *Research on Autonomous Vehicles at the Naval Postgraduate School*, Naval Research Reviews, Office of Naval Research, Washington, D.C., Volume XLIV, Number 1, Spring 1992.

9. Byrnes, R. B., *The Rational Behavior Model: A Multi-Paradigm, Tri-level Software Architecture for the Control of Autonomous Vehicles*, PhD Dissertation, Naval Postgraduate School, Monterey, California, March 1993.

10. Thornton, F., *A Concurrent, Object-Based Implementation of the Rational Behavior Model*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1993.

11. Brutzman, Donald P., *A Virtual World for an Autonomous Undersea Vehicle*, Ph.D. Dissertation, Naval Postgraduate School, Monterey, California, December 1994.

12. Kwak, S. H., McGhee, R. B., and Bihari, T.E., *Rational Behavior Model: A Tri-level Multiple Paradigm Architecture for Robot Vehicle Control*, Technical Report NPSCS-92-003, Naval Postgraduate School, Monterey, California, March 1992

13. Dijkstra, E. W., *The Humble Programmer*,Communications of the ACM, Volume 15, Number 10, p. 861, October 1972.

14. Ross, D. T., Goodenough, J. B., and Irvine, C.A. *Software Engineering: Process, Principles, and Goals*, IEEE Computer, Volume 18, Number 5, pp.65, May 1975.

15. Fisher, D. A., *A Common Programming Language for the Department of Defense -- Background and Technical Requirements*, Institute for Defense Analysis Report P-1191, June 1976.

16. Luqi, Berzins, V. and Yeh, R., *A Prototyping Language for Real-Time Software*, IEEE Transactions on Software Engineering, Volume14, Number 10, pp.1409-1423, October 1988.

17. Memorandum, Office of the Secretary of Defense, Subject: *Specification and Standards -- A New Way of Doing Business*, 29 June 1994.

18. American National Standards Institute Standard ANSI-1815A-1983.

19. International Organization for Standardization Standard ISO 8652-1987.

20. Federal Information Processing Standard FIPS 119.

21. Coste-Maniere, E., *A Synchronous/Asynchronous Approach to Robot Programming*, Proceedings of the Fifth Euromicro Workshop on Real-Time Systems, pp. 268-273, June 1993.

22. Harrison, A., and Moulding, M. R., *Object-oriented proramming languages for real-time rule-based systems: a practical evaluation*, Proceedings of the IEE Colloquium on Rule-Based Systems for Real-Time Planning and Control, Digest Number 160, Section 7, pp. 1-6, October 23, 1991.

23. Coffee, Peter, *Subtract C, Add Ada: Results Multiply*, PC Week, December 1994.

24. DoD Instruction 5000.2, Part 6, Section D (Computer Resources): Policy: Programming Languages.

25. Memorandum, Office of the Secretary of Defense, Subject: *Use of Ada*, 26 August 1994.

26. Public Law 102-396, Section 9070.

27. Constance, Paul, *Survey confirms Ada is top DOD language*, Government Computer News Volume 14, Number 9, p.1(2), May 1, 1995.

28. Taft, S. T., *The Ada 95 Philosophy,* Journal of Object Oriented programming, Volume 34, Number 6, pp.6 -8, June 1995.

29. Autonomous Mobile Robots: Control, Planning, and Architecture, S. S. Iyengar and A. Elfes, eds., IEEE Computer Society, 1991.

30. Elbert, T. F., *Embedded Programming in Ada*, Van Nostrand Reinhold, New York, p. 41, 1986.

31. Power, Kevin, *Mandate is to ensure software will fly,* Government Computer News, Volume 14, Number 6, p. 24(1), March 20, 1995 .

32. Endoso, Joyce, *Sounding DOD's sea of languages*, Government Computer News, Volume 13, Number 26, p. 53, Dec 12, 1994.

33. Chun, R., R. Lichota, B. Perry, and N. Sabha, *Synthesis of parallel Ada code from a knowledge base of rules*, Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing, pp. 600-607, December 2-5, 1991.

34. Coste-Maniere, E., Chun, B. Espiau, and D. Simon, *Reactive Objects in a Task Level Open Controller,* Proceedings of the 1992 IEEE International Conference on Robotics and Automation, pp. 2732-2737, May 1992.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   Cameron Station
   Alexandria, VA 22304-6145

2. Library, Code 013 2
   Naval Postgraduate School
   Monterey, CA 93943-5101

3. Computer Technology Programs, Code CS 1
   Naval Postgraduate School
   Monterey, CA 93943-5000

4. Dr. Ted Lewis, Code CS/Lt 2
   Chair, Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5100

5. Dr. Robert McGhee, Code CS/Mz 2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5100

6. Dr. Man-Tak Shing, Code CS/Sh 2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5100

7. Dr. Anthony J. Healey, Code ME/Hy 2
   Mechanical Engineering Department
   Naval Postgraduate School
   Monterey, CA 93943-5100

8. Lieutenant Colonel Ronald B. Byrnes, U. S. Army, Ph.D.. 2
   Senior Computer Scientist, Software Technology Branch
   Army Research Laboratory
   115 O'Keefe Building
   Georgia Institute of Technology
   Atlanta, GA 33032-0862

9. Dr. Donald P. Brutzman, Code UW/Br 1
Undersea Warfare Academic Group
Naval Postgraduate School
Monterey, CA 93943-5100

10. Dr. Luqi, Code CS/Lq. 1
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5100

11. Dr. James Eagle, Code UW 1
Chair, Undersea Warfare Academic Group
Naval Postgraduate School
Monterey, CA 93943-5100

12. CDR Michael J. Holden, USN, Code CS/Hm 2
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5100

13. Colonel Frederick I. Holden, USA (Ret.) 1
1505 North Lake Mirror Drive
Winter Haven, FL 33881

14. David Marco, Code ME/Ma 1
Mechanical Engineering Department
Naval Postgraduate School
Monterey, CA 93943-5100

15. Dr. Brian S. Bourgeois, Code 7442 1
Electronics/System Engineer, Advanced Sensor and Survey
Naval Research Laboratory, Mapping, Charting and Geodesy Branch
Stennis Space Center, MI 29529-5004

16. Michael Lee 1
Senior Research Engineer
Monterey Bay Aquarium Research Institute (MBARI)
160 Central Avenue
Pacific Grove, CA 93950